2

Report No. 7171

AD-A227 648

A Rate-Based Congestion Control Algorithm for the SURAP 4 Packet Radio Architecture (SRNTN-72)

Julio Escobar, Gregory Lauer, and Martha Steenstrup

1990

Prepared By:

BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02138

Prepared for:

DARPA/ISTO
1400 Wilson Bl.
Arlington, VA. 22209

Sponsored by:

The Defense Advanced Research Projects Agency
Under Contract No MDA-903-83-C-0173

DTIC
S ELECTE D
OCT 10 1990
B

The views and conclusions contained in this document are those of the authors and do not represent the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the Army or the United States Government

90 10 09 166

~~90 [illegible] 20 167~~

Abstract

We present a distributed, closed-loop, rate-based congestion control algorithm for regulating traffic flow rates in the SURAN packet radio environment. The algorithm's usefulness is not limited to SURAN, however: our congestion control algorithm could be tailored to function in arbitrary packet radio networks. Each packet radio periodically computes the maximum rate or *ration* at which a single traffic flow may use the node with minimal risk of congestion. The ration is based upon the desired versus measured load on node-associated resources, including the transceiver and the buffers. Each source packet radio controls its traffic submission rate to each destination according to a flow ration, which is the minimum of the rations computed by the nodes along the path.

The proposed congestion control algorithm effectively regulates source flows without severely limiting network throughput, accommodates a variety of network traffic patterns, incurs minimal overhead, and admits a simple implementation. The algorithm achieves max-min fairness in a large class of situations; however, channel access interaction among radios can pose problems in guaranteeing max-min fairness. We present an extensive set of analysis and simulation results describing the performance of the algorithm under a variety of conditions. (A+)

DTIC
COPY
INSPECTED
4

Accession For
NTIS GRA&I ☑
DTIC TAB □
Unannounced □
Justification

By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Contents

1 INTRODUCTION **2**

1.1 APPROACHES TO CONGESTION CONTROL 2

1.2 CONGESTION CONTROL ALGORITHM DESIGN GOALS 3

1.3 CONGESTION CONTROL ISSUES IN SURAN 4

1.4 DOCUMENT CONTENTS 6

2 LINK PROTOCOLS **7**

2.1 ARCHITECTURE 7

2.2 CHANNEL ACCESS PROTOCOLS 8

2.3 LINK FLOW CONTROL 9

2.4 PARAMETER SELECTION ALGORITHM 9

3 CONGESTION CONTROL ALGORITHM **11**

3.1 OVERVIEW 11

3.2 RESOURCE RATION COMPUTATION 11

3.2.1 Convergence 12

3.2.2 Responsiveness 13

3.2.3 Fairness 13

3.2.4 Robustness 13

3.2.5 Stability 14

3.3 PACKET RADIO RESOURCES 15

3.3.1 Transceiver Load 16

3.3.2 Processor Load 18

3.3.3 Output Link Throughput 19

3.3.4 Buffer Use 20

3.4 RATION DISTRIBUTION 21

3.4.1 Flooding 21

3.4.2 Tagging 21

3.4.3 Backpropagation 22

3.5 RATION ENFORCEMENT 22

4 PERFORMANCE RESULTS **23**

4.1 INTRODUCTION 23

4.2 SIMULATOR 24

4.3 THROUGHPUT AND DELAY 25

4.4 BOTTLENECKS 34

4.4.1 Contention Model 34

4.4.2 Queuing Model 35

4.4.3 CPU 36

4.4.4 Output Link Throughput . . . 45
4.5 FAIRNESS . . . 53
4.5.1 Transceiver Driven Rations . . . 54
4.5.2 Buffer Driven Rations . . . 66
4.5.3 Alternative Schemes . . . 74
4.6 CONVERGENCE . . . 77
4.7 ROBUSTNESS . . . 79

5 CONCLUSIONS **81**
5.1 SUMMARY . . . 81
5.2 CONCLUSIONS . . . 81
5.3 FURTHER RESEARCH . . . 83

A NETWORK PARAMETER VALUES **84**

1. INTRODUCTION

In this document, we present the research and design of a congestion control algorithm for the Survivable Adaptive Networking (SURAN) packet radio project. The work described here has been sponsored by the Defense Advanced Research Projects Agency (DARPA) and has been carried out at BBN. The proposed congestion control algorithm is designed to operate in the SURAN protocol suite number four (SURAP 4). This suite of protocols or packet radio architecture, is designed to support up to thousands of mobile nodes using hierarchical routing. In the SURAN environment, packet radios are known as *low-cost packet radios* (LPRs), a name that reflects a necessary goal for a network with thousands of nodes. We use the term "node" to refer to any packet switch in any packet-switching network, but we reserve the term "LPR" for the packet radio hardware used in the SURAN environment.

The SURAP 4 protocol suite employs receiver-directed spread spectrum multiple access (SSMA) transmissions for increased throughput; adaptive transmission power, FEC, and channel bit rate to adjust to changing radio link noise and jamming; and a security architecture to prevent or detect misbehaving nodes. SURAP 4 is an evolutionary step from earlier SURAP versions created and researched under the same DARPA contract. An overview of the SURAP 4 architecture can be found in [5].

Network congestion – the phenomenon of increased delay and reduced throughput in response to an increase in offered traffic – is a potential problem in any packet-switching network where mismatched transmission rates or convergent flows are common. Two conditions must hold before congestion can occur: persistent, excessive demand on a server, and packet transport using a protocol that relies on retransmissions to facilitate data delivery. In the SURAN packet radio environment, the node-associated servers include the transceiver and the main protocol processor (Intel 8086).

The first condition, offered load in excess of server capacity, imparts delay to all flows attempting to gain access to the saturated server. Packets are forced to queue behind the server or are refused service altogether, depending upon buffer availability at the node. In this document, we use the term *offered load* to denote the total traffic demand on a server, regardless of whether this load is serviced or not. We reserve the term *carried load* to denote that portion of the offered load which successfully receives service.

The second condition, packet retransmission, lowers the effective throughput of the already-saturated server. Retransmissions of refused packets consume resources which could otherwise be used for initial transmissions of new packets, thus decreasing the efficiency of server use. Packet retransmissions can lead to buffer shortages in nodes upstream from the saturated server, thereby propagating congestion to other servers.

The performance degradation associated with congestion may be confined to only those traffic flows converging on a saturated server, but in the worst case, is experienced by all traffic flows. For those flows experiencing congestion, the performance degradation is more often catastrophic than gradual. In the worst case, a complete collapse of communication service results. To minimize or avoid the risk of the disabling effects of congestion, most packet-switching networks employ some kind of mechanism for exercising control over traffic flow rates.

1.1 APPROACHES TO CONGESTION CONTROL

The prevalence of the congestion problem in packet-switching networks has elicited the design of many different congestion control strategies. There are six basic approaches to curtailing or preventing congestion: withhold excess traffic at the network periphery, reserve network resources in advance, add resources to the network, route traffic around congestion, spread traffic over multiple paths, and drop excess traffic in the network.

To control congestion in the SURAN packet radio environment, we recommend a mechanism for preventing excess traffic from entering the network. This approach is effective and easy to implement. Below, we describe the other approaches and the reasons that they were not selected. In Section 3, we present a detailed description of the proposed SURAP 4 congestion control algorithm.

Resource reservation works by allocating network resources necessary to handle a given end-to-end traffic flow. This approach requires – at each intermediate node between source and destination – a flow setup stage to reserve the requested resources prior to flow initiation and a flow teardown stage to release the resources after flow cessation. The principal difficulty in designing a resource reservation algorithm is how to achieve fair resource allocation among competing flows when resource capacity is not sufficient to satisfy the requests. Resource reservation does not encourage efficient use of network resources: unused reserved capacity can only be used by the reserving flow.

Adding network resources postpones but does not control congestion. There are two types of resources that can be added to the network: server capacity, in the form of transmission and processing bandwidth, and node memory, in the form of buffers. One may choose to add excess server capacity where congestion is most likely to occur. However, our past experience has shown that, in general, user demand expands to fill network capacity. Moreover, providing additional capacity is not always possible. Geographical location of nodes and budgetary constraints may restrict the quantity and quality of transmission service available. One may also choose to provide additional node buffers for absorbing short bursts of traffic. However, additional buffering has two disadvantages. It increases the mean and variance of packet-delay, and it retards the application of control on the traffic sources.

Selectively routing traffic around saturated servers is a form of congestion control. As a server becomes saturated, end-to-end flows select alternate routes that do not include the server. However, this approach has two disadvantages. First, it is sluggish to respond to congestion. Second, if alternate routes have scarce resources, the method simply moves the site of congestion from one location to another, rather than eliminating it.

Multiple routes between a given source and destination offer additional traffic-carrying capacity. By spreading source traffic over multiple paths, one reduces the chance that a single source flow will cause congestion. The use of multiple-path routing improves network throughput for any given user and thus raises the limit of network loading at which congestion is expected to occur. We strongly advocate the incorporation of a multiple-path routing algorithm into the SURAP architecture. Our proposed congestion control algorithm will work with single-path or multiple-path routing.

Dropping packets at a saturated server eliminates the formation of lengthy queues, and hence leaves buffers available for packets using other servers. However, the retransmissions necessary to force packets through to the destination may waste network resources that could otherwise be devoted to traffic that does not flow through the saturated server. The SURAP link-layer data transmission protocol already drops packets as a last resort measure.

1.2 CONGESTION CONTROL ALGORITHM DESIGN GOALS

The following goals guided our design of the congestion control algorithm for the SURAN packet radio environment. We note, however, that we consider these goals relevant to the design of a congestion control algorithm for any packet-switching network, including packet radio networks, wire-based networks, or internetworks.

- The algorithm should regulate user traffic flows subject to the conflicting constraints of minimizing congestion and maximizing throughput.
- The algorithm should be fair, granting equal access to all traffic flows converging on a server, subject to the constraint of traffic precedence.
- The algorithm should be stable under constant offered load and responsive to changing offered load.
- The algorithm should provide effective congestion control over a wide range of traffic patterns and network topologies

- The algorithm should be robust, capable of performing well under stress caused by node failures or external interference.
- The algorithm should interact synergistically with routing and with the link-layer protocols
- The algorithm should be efficient, consuming minimal amounts of transmission capacity and node processor cycles and memory.
- The algorithm should be easy to implement and deploy.

In designing a congestion control algorithm for the SURAN environment, we examined the following alternative approaches: anticipatory versus reactive strategies, open-loop versus closed-loop control, distributed versus centralized control, and local versus global control information. Ultimately, we settled on an anticipatory congestion control algorithm that employs closed-loop distributed control based upon global information.

An *anticipatory* congestion control strategy attempts to prevent congestion by reducing offered load in anticipation of congestion, whereas a *reactive* congestion control strategy attempts to alleviate congestion by reducing offered load in response to congestion. Anticipatory strategies rely on certain performance measures to predict congestion before it occurs. To determine the performance measures that are accurate predictors of congestion, one must have accurate models of traffic patterns and network behavior in response to these patterns. Reactive strategies are much simpler to design because symptoms of congestion are much easier to identify than predictors of congestion. However, reactive strategies can only eliminate congestion after it has occurred. In this case, many flows may experience a substantial increase in delay and reduction in throughput before congestion control becomes active. Reducing traffic rates in anticipation of rather than in response to congestion serves to prevent congestion altogether rather than to control it once it occurs.

Open-loop congestion control sets a single rate for a given source-destination traffic flow and does not adjust this rate in response to measured performance, whereas closed-loop congestion control continually regulates a flow's rate in response to measured performance. *Closed-loop* congestion control works well if feedback delay is small compared to the interval between fluctuations in offered load, and if congestion lingering during feedback delay affects only a small quantity of traffic.

Some aspects of congestion control lend themselves to a distributed implementation. In particular, each traffic source should have the ability to regulate its own flows, and each node should have the ability to measure its own performance. However, computation of the control signals governing flow regulation can be distributed throughout the network or centralized at one location. *Distributed* congestion control has several advantages. It is less susceptible to single points of failure, it is usually more responsive, and it can be installed in the nodes themselves. *Centralized* congestion control is preferable only when a large amount of memory or processing power is required to compute the control signals.

If a traffic source must rely on *local* information, that is, the degree of congestion in proximate servers, in order to reduce its flow rate in response to congestion, then it must wait until the effects of congestion propagate back to its vicinity. In the meantime, network performance may be severely impaired. Using *global* information, that is, the degree of congestion in distant as well as in proximate servers, a traffic source can more quickly and more accurately reduce its flow rate in response to congestion at another location in the network.

1.3 CONGESTION CONTROL ISSUES IN SURAN

In SURAP 3, the predecessor of SURAP 4, there is no explicit congestion control algorithm; however, there are a number of flow control mechanisms for regulating traffic in the network. Flow control is exercised at the transport layer by means of TCP windows and at the link-layer by means of the *pacing* algorithm, *single threading*, and *fairness queueing*. All of these flow control mechanisms do help to alleviate congestion; however, none but the last makes any attempt to achieve fairness among competing flows using a given server. A description of these algorithms can be found in [4].

The objective of the *pacing algorithm* is to restrict a node's average packet transmission rate to a neighbor so as not to exceed the rate at which the neighbor can relay the received packets. With pacing, a node uses a running estimate of the average packet forwarding delay of its downstream neighbor in order to control its rate of transmission – both first transmissions as well as retransmissions – to that neighbor. This pacing action is implemented by enforcing a *pacing delay* for all transmissions to that neighbor.

Forwarding delay is a somewhat problematic metric on which to base flow control decisions. First, delay in receiving a link-level acknowledgement can be caused by any of the following: queueing delay at a busy node, buffer shortages, bit errors in transmission, or a next-hop transceiver serving another packet (blocking). Delay resulting from heavy load should be responded to by reducing the packet transmission rate, whereas delay resulting from packet corruption shouid not. In order to choose the best transmission rate, a node must know the conditions under which forwarding delay is measured, but this is not possible in SURAP 3. Second, even if an increase in forwarding delay is the result of heavy traffic load, it is not possible under SURAP 3 for a node to determine which of the end-to-end traffic flows it carries, if any, is a major contributor to the heavy traffic load. A node measuring high forwarding delay to a neighbor will unconditionally restrict its packet transmission rate, using the same restrictive pacing delay for all its flows to that neighbor, regardless of the size of its traffic contribution to the congestion. Hence, flows not contributing to congestion may be unnecessarily restricted.

Single threading means that a node can buffer at most one unacknowledged packet at a time from each neighbor. The advantages of this mechanism are that it prevents any one neighbor's traffic from monopolizing the buffers in a given node, and it quickly backs excess demand into the source hosts. However, single threading also has some disadvantages. First, it does not permit a node to buffer a burst of traffic from a neighbor and hence encourages retransmissions. Second, it may hold back flows that are not causing congestion. For example, suppose that node A sends traffic to nodes C and D through node B, and that C is not a busy node but D is. Assume that many flows pass through D so that there is significant contention for D's transceiver. Hence, packets travelling from B to D will usually require retransmission and hence will have a long residence time in node B. With single threading, while an unacknowledged packet from A bound for D is waiting in B, no packets from A bound for C will be able to enter B. Hence, non-congesting traffic may be unnecessarily restricted.

The *fairness queueing* discipline applies to the output transmission queue at each node, and is essentially equivalent to round-robin scheduling of packet transmissions to neighbors. Fairness queueing ensures that traffic flowing to one neighbor does not monopolize the transmission capacity of the transceiver at the expense of traffic flowing to other neighbors. However, it does not ensure fair allocation of transmission capacity among different flows travelling to the same neighbor.

At the TCP level, one can achieve a degree of congestion control with Jacobson's dynamic window sizing algorithm. With this algorithm, congestion is controlled by a dynamic window, called the *congestion window* associated with each TCP connection. The window size depends upon the state of the TCP connection and the congestion perceived between source and destination. When a connection starts or resets, the host initializes the congestion window size to one packet, and on each subsequent TCP acknowledgment received increases the window by one packet – a s rategy known as *slow start*. A TCP retransmission timeout is interpreted by the host as an increase in network round-trip delay and thus as an indication of congestion. Following a timeout, the host reduces the congestion window size by a fractional multiplicative factor, 0.5. When an acknowledgement is received, the host increases the congestion window size by an additive factor, one packet, as at connection initialization.

However, these TCP enhancements do not solve the fairness problem. The slow start algorithm may unnecessarily restrict throughput, since it does not have knowledge of network capacity when setting initial flow rates. Short duration flows, in particular, are most affected by the artificial throughput reduction. When congestion spreads, flows may experience long delays even though they are not the cause of the original congestion. Hosts reduce the congestion windows for all flows experiencing delay, not just those causing congestion. Hence, some traffic may be unfairly restricted.

For SURAP 4, we propose a distributed, rate-based congestion control mechanism applied to end-to-end traffic flows at their sources, which uses global information about the state of network congestion to determine traffic rates. Each node computes a maximum flow rate or *ration* based upon measured versus desired load on node-associated resources. All flows using a given resource are assigned the same ration, thus encouraging fairness between competing flows. Each source node computes a flow ration to each destination as the minimum of the rations for nodes along the path. The flow ration controls the rate at which traffic is submitted by the source to the destination.

1.4 DOCUMENT CONTENTS

In this document, we describe the proposed congestion control algorithm for SURAP 4, the principles behind the design, and the performance results obtained through analysis and computer simulation. In Section 2, we review the link-layer protocols of SURAP 4 and describe how they interact with congestion control. In Section 3, we discuss the proposed algorithm in detail and the issues involved in designing a congestion control algorithm for packet radio. In Section 4, we present congestion control performance results obtained by analysis and computer simulation. In Section 5, we summarize the main conclusions and suggest future research directions.

2. LINK PROTOCOLS

2.1 ARCHITECTURE

The link protocols are the protocols regulating packet access to the transmission media and storage at the nodes. A notion of the link protocols in the SURAP 4 architecture is necessary to understand the operation of the congestion control algorithm. In this section, we give a simplified overview of the algorithms involved. These protocols are documented in detail in [6,7,3]. Algorithms addressing network control do not critically affect the performance of the congestion control algorithm unless they fail to keep the network functional.

The SURAP 4 architecture uses *receiver directed, direct sequence spread spectrum* transmission. This means that a packet is encoded with a pseudorandom sequence recognizable only by its intended receiver radio. The throughput improvements obtained with this CDMA scheme are documented in [8]. There is also a *broadcast* sequence which is recognizable by all radios, providing a multicast capability. We divide the link protocols, or algorithms, into *channel access* protocols, *link flow control* protocols, and *physical layer* protocols.

Channel Access. In our broadcast medium, channel access is regulated by four protocols.

- Spacing Algorithm;
- Moment of Silence Algorithm;
- Retransmission Protocol;
- Alternate Routing Protocol;

The *spacing algorithm* determines a minimum interval between packets transmitted on a link. The intention is to avoid sending a new packet before the neighbor is ready for it. The spacing interval is typically short and of low variance. The *moment of silence algorithm* gives acknowledgements high priority in the network so that network resources can quickly be freed and reused. The *retransmission protocol* regulates retransmission intervals for unacknowledged data packets. It also limits to six the maximum number of packet transmission attempts before packet discard. Retransmission intervals are typically higher than, but of the same order as, spacing intervals. The *alternate routing protocol* specifies that after three failed transmission attempts, a packet should be broadcast using the broadcast spread spectrum sequence. Thus, any radio within range can help in forwarding the packet to the next hop in its path to the final destination.

Link Flow Control. Link flow control is intenede to confine the congesting effects of high rate traffic. Two protocols enforce link flow control:

- Link Window Protocol;
- *K*-Buffering Protocol.

The *link window protocol* simply specifies that a new data packet can be sent to a given neighbor, only after that neighbor has acknowledged the previous data packet. This window mechanism prevents drowning neighbors with packets. The *K-buffering protocol* limits to K the number of packets that a node can buffer from a given neighbor, thus preventing single neighbors from monopolizing node buffers. It also imposes increasingly larger retransmission intervals for neighbors who have reached their K limit.

There is a trade off between the quick response to transients and local problems provided by the fast-acting link protocols and the global fairness and stability provided by the congestion control algorithm. In designing the link protocols, care was given to preventing link flow control and channel access algorithms from usurping the role of congestion control. By shielding a region from high-rate traffic converging on it, the link protocols can prevent the congestion control algorithm from detecting and controlling the source of congestion. It was therefore necessary to redesign the waiting intervals imposed by spacing, retransmissions, and especially K-buffering so that natural packet flow was only seriously impeded in extreme situations.

Physical Layer Protocols. We are only concerned with two protocols at the physical layer:

- Parameter Selection Algorithm;
- Link Up/Down Protocol.

We are not concerned with protocols determining packet format, modulation, and other physical layer functions. The *parameter selection algorithm* adaptively adjusts transmitter power, coding scheme, and channel bit rate from packet to packet in order to improve transmission quality. Its actions affect link throughput and forwarding delay, and hence indirectly affect congestion control. The *link up/down protocol* determines whether a link should be established or terminated.

2.2 CHANNEL ACCESS PROTOCOLS

In the receiver directed architecture proposed, all acknowledgements are active acknowledgements. A special acknowledgement packet is created by the receiving node and transmitted back to the sending node; as these active acknowledgements are of short length, the channel time reserved for them represents minimal overhead. The *moment of silence algorithm* seeks to give high priority to acknowledgement packets, thus improving throughput by allowing nodes to free resources quickly. After forwarding a data packet, the sending node refrains from transmitting for an interval of MOS time units. This interval is known as the "moment of silence" and is a network parameter equal to the time required for the receiving node to prepare and start transmitting the acknowledgement. The moment of silence at the sender can only be preempted by packet reception. The receiving node gives highest priority to preparing and transmitting the acknowledgement upon receiving a data packet, so as to comply with the moment of silence at the sender. Data transmissions or retransmissions are delayed until the acknowledgement is sent.

The *spacing algorithm* enforces a minimum time interval between successive first transmissions of data packets. The spacing interval allows for packet processing, packet transmission plus moment of silence, and acknowledgement reception at the receiving node. The interval is enforced from the end of an acknowledgement reception or the end of the moment of silence at the sending node, whichever comes first. Computation of the value of spacing interval is done at the receiving node and communicated back to the sender in the acknowledgement.[3]

Concurrently, the *retransmission protocol* sets a retransmission timer as soon as a packet transmission ends. If the acknowledgement for a transmission arrives before this timer expires, then any residual spacing wait is observed before transmission of the next data packet; otherwise, a second transmission is attempted and a new retransmission timer is set. If six transmissions fail, the buffer containing the packet becomes available for any other packet requiring a buffer. The timer w_{n+1} for scheduling transmission $n+1$ is computed according to

$$w_{n+1} = 2\ \mathrm{MOS} + n\,(\mathrm{MAX\_PKT\_TIME} + \mathrm{MOS})\ \mathrm{RANDOM}. \qquad (2.1)$$

[3] If the receiving node is the final destination for a packet, then the spacing value computed is smaller since there is no need to relay the packet.

where k is a small integer constant, chosen equal to 3 in our design. MAX_PKT_TIME is the longest duration of a packet on the channel (approximately 60 milliseconds with channel rate 100Kbps and with 1/2 FEC rate), n is the index of the most recent transmission, and RANDOM is a random variable uniformly distributed between 0 and 1. The above formula is similar to linear back-off retransmission strategies, where the retransmission interval increases with the number of retransmissions. Using MAX_PKT_TIME is a heuristic way of accounting for large packets on the channel, while the randomness prevents synchronization among retransmitting nodes. A special retransmission interval applies in the event of bit errors detecting upon packet reception. The receiving node acknowledges the packet through an *error acknowledgement*. The transmitting node uses a retransmission interval equal to the one in (2.1) minus the first term.

The *alternate routing protocol* exists to enhance the survivability of the network. It provides the capability to route around failed or ill-functioning nodes and links without invocation of the routing algorithm. After three failed transmission attempts, a node uses the broadcast spread spectrum sequence to send the packet. Radios within range that receive this packet and that are at least as close to the destination as the sender are allowed to acknowledge the packet and forward it towards its final destination. With alternate routing, the packet's ultimate path to the destination is unknown. However, our congestion control algorithm is designed to control traffic flow rates along known paths. We describe how to resolve this dichotomy in Section 3.4, where we discuss how congestion control information is propagated throughout the network.

2. LINK FLOW CONTROL

The *link window protocol* simply specifies a link window size of one. An acknowledgement must be received for a given packet before the next one can be forwarded. Larger window sizes were rejected for now, because of the limited buffering resources in the nodes and the relative reliability of acknowledgement reception due to the *moment of silence algorithm. The subject of optimal link window size in SURAP 4 has not been studied in detail.*

K-Buffering regulates the use of buffers in a node. A neighbor relaying traffic through a node is allowed at most K packet buffers in the node at any given time. A host attached to a node is allowed at most two buffers. If new packets from a neighbor about to reach its K limit are received ($K - 1$ buffers in use), the packets are not buffered. A special acknowledgement packet is returned to the neighbor, requesting it to observe an exponentially increasing waiting time for successive retransmissions, until one or more of its packets is forwarded. The waiting interval is based on a time-window averaged service time for packets from this neighbor, which is periodically computed at the relaying node, but which is never increased past a maximum value (1.5 seconds in our design). This flow control mechanism is far more drastic than spacing or retransmission intervals and is only invoked when the K limit is reached.

2.4 PARAMETER SELECTION ALGORITHM

Three channel parameters can be independently and adaptively adjusted, in discrete steps, from packet to packet in the SURAP network radio links. These are transmitter power, forward error correction (FEC) coding rate (convolutional codes), and channel bit rate. In the network design, we recommend using FEC for all transmissions, since the coding overhead incurred for the weakest coding rate is minimal (8/7) and the sequential decoder hardware makes available a "bit error count" by comparing received and decoded channel bits.

The *parameter selection algorithm* adaptively chooses the operating parameter triplet based on channel feedback after each transmission. Triplets are ranked in order of signal to noise ratio (SNR) gain. The greater the SNR gain of the triplet chosen, the greater the adverse impact on neighbors and on link throughput. The parameter selection algorithm chooses the triplet with minimum possible adverse impact subject to providing a "satisfactory" SNR gain. An SNR gain is deemed satisfactory if it is enough for a maximum length packet to maintain a packet error probability below a predesigned value (network parameter equal to 0.1) when transmitted over the link in the absence of contention. This aims at maintaining a fairly predictable behavior of the link[2] and providing a rational interaction with higher level protocols.

The bit error counts computed upon reception are included in packet acknowledgements for the use of the transmitting node. SNR gain on a link is decreased by a transmitting node after enough successful transmissions have been observed. The gain is increased due to failed transmissions or error acknowledgements. The bit error count is mainly used to estimate the change in SNR gain required for satisfactory link operation. The bit error counts, and a count of packets received with post-decoding errors[3], are used to estimate the packet error probability resulting from noise alone. This estimate is factored out from attempts-per-packet computations in the congestion control algorithm.

Relevant to the congestion control algorithm is the fact that the parameter selection algorithm keeps the number of retransmissions induced by noise at a fairly negligible level so that most of the retransmissions observed are due to contention (unless the channel is so noisy that, at maximum gain, there is still a significant possibility of packet corruption). Parameter selection also makes packet transmission times vary, possibly from packet to packet and among distinct links of the same node, in a way only partially dependent on packet length. This affects the service time of a link.

The *link up/down protocol* specifies rules for establishing and terminating a radio link. When two radios come within range of each other, they determine whether it is possible to establish a link. If so, link parameter establishment is handed over to the parameter selection algorithm. If a link operating at maximum SNR gain fails to forward a packet a given number of times, the link is terminated.

[2] Retransmissions are considered to be the principal factor in link delay.

[3] [illegible] errors are detected by checking the [illegible] which [illegible] small length is less prone to errors than the packet as a whole.

3. CONGESTION CONTROL ALGORITHM

3.1 OVERVIEW

The SURAP 4 congestion control algorithm is an anticipatory strategy enforced by a distributed, closed-loop control algorithm that uses information about resource load along paths to regulate the rates at which source flows enter a packet-switched network. Flows are identified by their source and destination packet radios, not by their source and destination hosts nor applications running on those hosts. Hence, data packets from multiple source hosts homed to a packet radio A sent to one or more destination hosts homed to a packet radio B constitute a single traffic flow between A and B.

Nodes within a network collectively play the principal role in controlling congestion. Each node periodically computes, for each of its associated resources, the ideal flow rate or *resource ration* based upon the desired versus offered load levels. In the SURAN environment, resources include the two servers, namely the transceiver and the main protocol processor (Intel 8086), as well as buffers and *output link throughput*, a concept we describe later on in Section 3.3.3. The resource ration is the maximum rate at which a single traffic flow may use the resource, with minimal risk of congestion, and depends upon the capacity of the resource, the number of flows using the resource, and the size of each flow. The single ration per resource permits each flow using a resource to have equal access to the resource.

Packet radios use resource rations to control traffic flow rates. Each source node computes its traffic submission rate or *flow ration* for each destination node as the minimum of the collected rations for component resources along the path to the destination. A source node enforces flow rations using *throttlers*, one per destination, that restrict the rate of traffic flow from the hosts to the store-and-forward network. Each throttler operates on a timer mechanism that determines when the next packet for the given destination can be released into the network.

The SURAN congestion control algorithm comprises three separate functions: ration computation, ration distribution, and ration enforcement, each of which we describe in detail in the following sections. Many aspects of this algorithm have been adapted from the congestion control algorithm developed by BBN [1] for wire-based networks such as the ARPAnet.

3.2 RESOURCE RATION COMPUTATION

The objective of resource ration control is to maintain the average traffic flow through a resource at a rate that neither saturates the resource nor severely restricts throughput. A packet radio controls a resource ration by monitoring the current load offered to the resource and by adjusting the resource ration according to the perceived traffic load.

We have chosen a load-based instead of a delay-based performance measure to affect ration control for the following three reasons. First, a congestion control strategy based on delay is reactive rather than anticipatory, and hence slower to recognize congestion. When a resource becomes saturated, packets queue behind the resource incurring delay. Using measured delay to control congestion means that congestion cannot be controlled until it has occurred, as delay is a symptom and not a predictor of congestion. Second, the high variance associated with delay measurements can result in instabilities, when delay is used to control flow rates. Third, it is difficult to design a fair congestion control strategy based on delay. Reducing rations according to delay measurements does not necessarily lead to fairness. Flows experiencing large delays are not necessarily sources of congestion, although they may be affected by congestion. Hence, by unconditionally restricting flows that experience delay, a node may unnecessarily limit non-congesting flows.

Resource rations control resource load, indirectly through flow rations which regulate source traffic flow rates through the resources along the source-destination paths. Each node periodically updates each of its resource rations according to the following ration update control law:

$$\begin{aligned} \mathit{ration}(0) &= \mathit{threshold}, \\ \mathit{ration}(t+1) &= \min\{\mathit{threshold}, \frac{\mathit{target}}{\mathit{load}}\mathit{ration}(t)\}, \end{aligned} \tag{3.1}$$

where *threshold* represents the maximum acceptable flow rate through the given resource, *target* represents the desired resource load level, and *load* represents the load offered to the resource. We refer to this ration update rule as the *basic* ration control law; in Section 3.2.5, we describe modifications of the basic law that appear necessary to stabilize rations in the SURAN packet radio environment.

Rations are expressed in either bits per second or packets per second, depending upon the resource with which they are associated. As we describe in Section 3.3, each node resource is predominantly affected by either bit or packet rate, but may be influenced by both. For simplicity, we have chosen to associate a single rate measure with a given resource. However, we have not gone so far as to assign a single unit – either bits or packets per second – to express traffic rates for all resources. Although such an approach provides the simplest representation of rations, it is not feasible in the SURAN environment for the following reason. To convert all rations to a single rate unit, one must use the average number of bits per packet. In the SURAN environment, packet sizes can vary between 100 and 6000 bits (including error correcting bits) in length. With such a large variance in packet size, it is not possible to achieve acceptable accuracy in representing those rations that require conversion to the common unit.

The objective of the resource ration control law is to maintain resource load at the target level. When the load offered exceeds the target, the node reduces the resource ration by the ratio of the desired load to offered load, referred to as the *ration adjustment ratio*. Similarly, when the load offered drops below the target, the node increases the resource ration by the ration adjustment ratio, provided the resulting ration does not exceed the threshold level. The *target* parameter in the ration control law serves two separate purposes. It acts as the resource load set point, and it influences the rate of ration adjustment. The *threshold* parameter serves to prevent rations from increasing without bound when traffic flows are light.

3.2.1 Convergence

A packet-switching network is a dynamic entity. The number and size of traffic flows and the routes over which they travel change with time. Under these conditions, a resource ration does not converge to a single value but fluctuates in response to changes in resource load. However, given constant-rate traffic sources, the resource ration should converge to the correct value by repeated application of the control law. The number of control law iterations required to achieve ration convergence depends upon the number of individual traffic flows using the resource and the load offered by each flow.

A resource ration at time t will converge to its proper value in one iteration of the basic control law at time $t + 1$, provided that all flows using the resource are *greedy* with respect to the larger of $\{\mathit{ration}(t), \mathit{ration}(t+1)\}$. (A greedy flow is one that always offers its full ration of traffic.) This condition is sufficient for single-step ration convergence for the following reason. The ration adjustment ratio $\frac{\mathit{target}}{\mathit{load}}$ is the correct factor by which to scale the combined rate of the flows using the resource. In fact, each flow's rate will be scaled by exactly this ratio, as all flows are greedy with respect to the larger of the rations computed at times t and $t + 1$. Thus, the combined offered load controlled by the newly computed ration will be equal to the target load level. If the greediness constraint is not met, ration convergence can require several applications of the control law, since the non-greedy flows will be scaled by less than the ration adjustment ratio. In the worst case, namely when the resource ration is at its maximum value and many non-greedy flows begin to use the resource, the number of control law iterations required to cause throttling is on the order of the logarithm of the number of flows using the resource.

3.2.2 Responsiveness

The congestion control algorithm must be responsive to significant changes in resource load so that congestion can be avoided and throughput maintained. Any delay incurred between detecting a change in load at a resource and updating the corresponding flow rations at the sources directly affects the algorithm's ability to adjust quickly to changes in load offered. There are three distinct sources of delay inherent in the congestion control algorithm. The first is the delay between the time that resource load changes and the time that the node updates the resource ration. The second is the delay between the time that the node updates the resource ration and time that the source nodes receive the updated information. The third is the delay between the time that the source nodes compute the flow rations based on the updated resource ration and the time that the flow rations takes effect. To promote responsiveness, ration computation, distribution, and enforcement schemes should impart minimal delay in the congestion control algorithm.

3.2.3 Fairness

A principal objective of the SURAN congestion control algorithm is to promote fairness among flows competing for a resource, specifically *max-min fairness*. The set of rates for traffic flows using a given resource is max-min fair if no flow rate in the set can be increased without forcing a decrease in another flow rate of equal or lower value.

Different flows are governed by different flow rations dependent upon their source-destination paths. Hence, not all flows using a given resource have the same flow ration. Nevertheless, the single ration value per resource restricts the maximum rate of each traffic flow using the resource to the same value and thus facilitates fairness among flows.

We note that in the presence of fluctuating traffic rates, a state of max-min fairness may not necessarily be attained. However, max-min fair flow control can be achieved *for constant-rate traffic sources, except when channel* access interaction among neighboring nodes results in efficient communication from one neighbor at the expense of inefficient (retransmissions) communication from another. This problem is discussed in detail in Section 4.5. In any case, congestion control rations should cause flow rates to move in the direction of max-min fair flows.

3.2.4 Robustness

The SURAP 4 congestion control algorithm must function acceptably in a mobile network environment. The algorithm will have time to react to resource loads and adjust flow rations accordingly, as long as paths persist for tens of seconds. Our congestion control algorithm relies on the assumption that source-destination paths fluctuate less frequently than the traffic load on them, and that they remain in existence long enough for ration control to adjust flow rates according to the available capacity of resources along the paths. The effectiveness of the congestion control algorithm decreases if network topology changes rapidly or if nodes make frequent use of alternate routing.

The SURAN congestion control algorithm must also function in the presence of passive component failure or intentional interference. When network components fail, the available network capacity decreases and the importance of efficient resource usage increases. Our congestion control algorithm facilitates efficient resource use by preventing congestion of the remaining resources and by fairly allocating capacity among the flows using these resources.

The SURAN congestion control algorithm must perform acceptably, even if certain nodes distribute erroneous ration information or fail to obey flow rations. When a node advertises artificially high rations, it runs the risk of becoming congested in the presence of heavy traffic. In this case, if the node does not reduce its rations, congestion will spread to its neighbors. The burden of congestion control falls to those nodes affected by the spreading congestion that do control their rations properly. Thus, congestion is contained and eliminated. When a node advertises artificially low rations, it can restrict the throughput of all flows passing through it. If a source node disobeys its flow rations, it can cause congestion along its paths. Moreover, it may force intermediate nodes along its paths to lower their rations and restrict the throughput of flows using them. Currently, there is no mechanism within the congestion control algorithm to resolve these problems. It is left to network security and management protocols to detect and correct problems involving misbehaving nodes.

3.2.5 Stability

When a resource is saturated and there is extensive queueing at preceding resources along the path, the node is unable to observe the effect of its ration reductions until the queues empty. If the node invokes the ration control law several times during this period, it may reduce the resource's ration below the correct point, before the queues empty. However, the corresponding reduction in throughput is temporary; the node takes corrective action, once it has measured low resource load.

Overshooting the target load level when decreasing a ration can be minimized by either of the following two methods. First, by properly configuring the number of buffers available for queueing packets at a node, one can limit the size of network queues and hence reduce the delay in observing the effects of ration reduction. Second, by properly choosing the length of the interval between successive applications of the ration control law, one can ensure that in most situations the node is able to observe the effect of its previous resource ration on the measured resource load level.

Our simulation results have shown that it is also possible for the ration to overshoot the target load level when increasing. In fact, we have observed that a large increase in a ration value can cause an explosive increase in the measured load resulting in oscillations of the resource ration about its proper value. The principal reason for overshooting on ration increase is a nonlinear relationship between the flow rate, governed by the ration, and the response, measured as "offered load". This nonlinearity occurs only when offered load is not expressed as a flow rate, but rather as some related performance measure. For two of a node's resources – the transceiver and the buffers – load is not a flow rate quantity. In particular, transceiver load is measured in terms of of contention, and buffer load is measured in terms of queue size. Both transceiver contention and queue length are known to exhibit markedly nonlinear responses to changes in traffic flow rate.

We have experimented with various modifications of the basic ration control law (equation (3.1)) that compensate for the nonlinear relationship that exists between flow rate and load for certain node resources. One of the most effective, referred to as the *asymmetric* ration control law, uses a fixed ration adjustment ratio for increasing resource ration values to bound the ration growth rate when measured load is low, and hence to bound the amplitude of the ration oscillations. The asymmetric control law is:

$$ration(t+1) \quad = \quad \min\{threshold,\ \frac{target}{load}ration(t),\ 1.2ration(t)\}, \tag{3.2}$$

where the asymmetry is in the rate of ration increase and ration decrease. Rations are reduced at the rate specified by the ration adjustment ratio, but rations are increased at a rate no greater than 1.2. The maximum ration increase rate of 1.2 was determined by simulation, as a compromise between oscillation elimination and fast convergence when rations must increase

We have also obtained promising results with a different modification of the basic ration control law, referred to as the *cube root* control law:

$$ration(t+1) = \min\{threshold,\ \sqrt[3]{\frac{target}{load}}ration(t)\}. \tag{3.3}$$

Taking the cube root of the ration adjustment ratio reduces the size of the steps between successively computed rations and helps to reduce oscillations, as we discuss in Section 4.6. In fact, applying this law to buffer rations and computing buffer use according to the number of neighbors (see Section 3.3.4) appears to completely eliminate steady state oscillations in situations where buffers are the bottleneck resource

3.3 PACKET RADIO RESOURCES

For the SURAN environment, we have selected the following set of node resources on which to impose congestion control rations: the transceiver, the main processor, the set of buffers, and the output links. We say that a *link* exists between any two nodes that communicate directly at the link layer. The choice of these particular resources was suggested by the following model of packet radio communication. Each packet transmitted by a node must contend for transceiver capture at the receiving node; following capture, the packet must await processing and then transmission to the next node on its path.

The resources associated with a node are categorized according to whether they represent capacity of the node or capacity of the link and whether their throughput is governed predominantly by bit rate or by packet rate. Node capacity reflects the limitations on the rate of data transmission to a node imposed by contention for the transceiver, by finite buffer space in the node, and by processing delays necessary for relaying a received packet. Link capacity reflects the limitation on the rate of data transmission from a node imposed by the quality of the link, which is affected by external factors such as physical obstacles, multipath fading, and signal interference (including jamming), and by the delays demanded by the link-layer protocols in the absence of contention for resources at the receiving node. The effect of this contention, when it exists, is captured in the capacity of the receiving node. This approach was adopted to prevent neighboring nodes from taking over the congestion control responsibility from the bottleneck that should perceive congestion.

Transceiver, buffer, and processor rations have significance on a node basis, whereas output link throughput has significance on a link basis. We define *output link throughput* as the maximum number of packets of a given size that can be transmitted across a link of the specified error and link-layer protocol characteristics. Link rations are necessary to accommodate link-specific behavior. Otherwise, a low-throughput link could not be individually targeted for flow control. Backlogs produced by such a link could only be controlled through node rations. This control would unnecessarily throttle other flows through the node which do not use the low-throughput link, resulting in unfair flow control.

Technically speaking, one cannot cleanly divide the node resources into two disjoint subsets, such that those in one subset are only affected by bit rate and those in the other subset are only affected by packet rate. For example, transceiver load is predominantly affected by bit rate. However, each data packet successfully received forces an acknowledgement which has a small but non-negligible effect on the bit rate, and thus packet rate affects transceiver load as well. A dual example exists for processor load, which is predominantly affected by packet rate. Forward error correction performed by the processor depends on the size of the packet, and thus bit rate affects processor load also. Nevertheless, for simplicity, we have settled on classification of resources according to bit rate or packet rate influence, since for each node resource one of the two is dominant. The transceiver is most affected by bit rate, whereas the processor, the buffers, and the output link throughput are most affected by packet rate.

In order to determine the proper target load level for each resource, one must have an accurate model of the throughput of a node. However, characterizing the throughput of a node is not a simple matter. For example, consider a node that receives data from two neighbors, such that the average packet size is the same for each neighbor. In this case, the highest throughput is achieved when most traffic is transmitted from one neighbor rather than when equal amounts of traffic are transmitted from both neighbors. The reason is that in the limit, there is only a single traffic stream, which results in synchronized packet arrivals and hence reduced contention for the transceiver. Throughput depends upon the characteristics of the traffic flowing through the node, which are determined by the number of communicating neighbors, the size of the traffic flows, the length of the packets, the channel error characteristics, and the link-layer protocols in use. We have yet to produce a satisfactory model of packet radio behavior that captures a wide variety of traffic patterns. However, through simulation, we have gained some insight into the behavior of the nodes under different conditions, and it is this knowledge that we have used in selecting target values for the given resource rations.

The resource rations computed by a node are based upon the resource load measured during the preceding *measurement interval*. To measure resource load, the node obtains a set of sample measurements and averages them uniformly over the measurement interval. Selecting an appropriate measurement interval duration involves tradeoffs between stability and responsiveness. A longer measurement interval smooths out transients in the load measurements, yielding an average resource load value that is a good predictor of longer-term use; however, it limits the speed at which actual changes in load can be captured in the measurement. A shorter measurement interval yields an average resource load value that is accurate in the shorter-term, allowing the ration control law to track fluctuations in load; however, it is unable to ignore transients.

For each resource, the measurement interval is identical to the interval between successive computations of the resource ration. With the exception of buffers, all resource loads are averaged over the same measurement interval. In our simulations, we experimented with measurement intervals ranging from two to five seconds in length. It is necessary to set the measurement interval long enough to collect a sufficiently large sample of packets on which to base resource load measurements. From the simulation results, we found that a measurement interval of three seconds in duratio: appears to be long enough to collect an acceptable number of samples and to enable the node to observe the effects of its previous ration values on the load on its resources. Buffer use is averaged over a longer period than measured load for the other resources, in an effort to smooth out the effects of the high variance between individual samples.

An alternative approach to using a fixed measurement interval is to use a fixed number of packet arrivals to determine the length of the measurement interval. Thus, the duration of the measurement interval will increase or decrease depending on the rate of packet flow. This scheme has the advantage of speeding up measurement computations when congestion is most likely, but it has the disadvantage of slowing down measurement computations when a large quantity of bandwidth suddenly becomes free. Although we have not simulated this scheme, we feel that it is promising and deserves further research. Also, It is worthwhile to research a combination of the two measurement methods discussed.

3.3.1 *Transceiver Load*

Our simulation results have shown that the transceiver is the most sensitive to traffic load of all of the node's resources. Hence, it is important for the node to accurately monitor transceiver load for ration computation. Lacking a tractable model of transceiver behavior that can easily be implemented in the nodes, we have selected the average number of packet transmissions to a node, as a reasonable indicator of transceiver load. The calculation of the average number of packet transmissions factors out the effect of link quality and hence the retransmissions resulting from packets containing bit errors and ignores those retransmissions resulting from inadequate buffering; only the effects of transceiver contention are captured.

Each node uses the average number of transmissions per packet due to contention in order to update its bits-per-second ration, since the amount of time that a tranceiver is busy is directly proportional to the number of bits received and transmitted. However, transmissions per packet cannot be converted directly to absolute bit rates, because very different traffic characteristics can result in the same average number of transmissions per packet. Nevertheless, the average number of transmissions per packet is an excellent indicator of transceiver activity.

A node calculates the average number of transmissions per packet as follows. Each packet carries in its header the number of the current transmission. Whenever a packet is successfully captured by the transceiver, deemed acceptable by the error detection/correction mechanism, and considered a new packet and not a duplicate by the link-layer data transmission protocol, the node reads the transmission count from the packet's header to determine the number of attempts required to obtain the successful transmission. Note that rejection of packets because of buffer shortages occurs after this point in packet processing. Thus, the transmissions per packet obtained from the packet header reflects a combination of the effects of link quality and contention for the transceiver.

Each node maintains a running sum $T_j(t)$ of transmission attempts and a running sum $P_j(t)$ of unique packets received per incoming link j during a given measurement interval t. The average number of transmissions per packet $A_j(t)$ on incoming link j is equal to

$$A_j(t) = \frac{T_j(t)}{P_j(t)}$$

during that measurement interval.

To extract the transmissions per packet resulting only from transceiver contention, the node must account for the error characteristics of each link. Let e_j be the probability that a packet arriving on link j contains bit errors. Let b_j be the probability that a packet arriving on link j is blocked at the transceiver. Then, the probability s_j that a packet is successfully received on link j is equal to $(1 - e_j)(1 - b_j)$. If a node were allowed an infinite number of transmissions per packet, then for a given incoming link j, the expected value of X_j – the number of transmissions per packet – would be equal to

$$E[X_j] = s_j \sum_{i=1}^{\infty} i(1 - s_j)^{i-1} = \frac{1}{s_j},$$

and the expected value of C_j – the number of transmissions per packet due to contention only – would be equal to

$$E[C_j] = \frac{1}{1 - b_j} = (1 - e_j)E[X_j].$$

The SURAP 4 link-layer data transmission protocol allots each data packet a maximum of six transmissions in which to achieve successful reception at the neighbor; after six unsuccessful transmissions, the node discards the packet. The transmission attempt samples gathered by the node are for those packets successfully received within six transmissions. Hence, $s_j(1 - s_j)^{i-1}$ – the probability that a packet requires exactly i transmissions for $i \le 6$ – must be normalized by $1 - (1 - s_j)^6$ – the probability that a packet requires six or fewer transmissions. Thus, $E[X_j]$ – the expected number of transmissions per packet – becomes

$$\begin{aligned} E[X_j] &= \frac{s_j \sum_{i=1}^{6} i(1 - s_j)^{i-1}}{1 - (1 - s_j)^6} \\ &= \frac{\frac{1-(1-s_j)^7}{s_j} - 7(1 - s_j)^6}{1 - (1 - s_j)^6} \\ &= \frac{1}{s_j} - \frac{6(1 - s_j)^6}{1 - (1 - s_j)^6}. \end{aligned}$$

Our measure of transceiver load is the number of transmissions per packet resulting from contention only, averaged over all links and all packets received within a given measurement interval. To compute the expected number of transmissions per packet due to contention on a single link j, one could start with the formula for the expected number of transmissions per packet, equate it to $A_j(t)$ – the sample average number of transmissions per packet, substitute e_j – the packet error probability computed using the bit error rate supplied by the parameter selection algorithm together with the measured average packet size – into the expectation formula, solve for b_j – the probability of blocking, and recompute the expected number of transmissions per packet substituting $1 - b_j$ for s_j. However, the complexity of this computation prohibits its use in the node. Instead, we have chosen the approximation $E[C_j] \approx (1 - e_j)A_j$, because it is easy to compute and yields acceptable values, although it tends to underestimate slightly the number of transmissions attributable to contention.

To determine the transceiver load in terms of the number of transmissions per packet caused by contention only, averaged over all links at a node during a given measurement interval t, the node need only compute:

$$E[C] = \frac{\sum_j P_j(t)E[C_j]}{\sum_j P_j(t)}$$

$$\approx \frac{\sum_i P_i(t)(1-\epsilon_i)A_i(t)}{\sum_i P_i(t)}$$

$$= \frac{\sum_i P_i(t)(1-\epsilon_i)\frac{T_i(t)}{P_i(t)}}{\sum_i P_i(t)}$$

$$= \frac{\sum_i (1-\epsilon_i)T_i(t)}{\sum_i P_i(t)}.$$

The congestion control algorithm depends upon the parameter selection algorithm to supply accurate bit error rates in order to calculate packet error probability estimates and in turn to compute transceiver load. Prior to SURAP 4, the link quality assessment algorithm was not able to distinguish between error-prone links and congested links, since it relied only on a comparison of the number of packets sent and the number of packets received. Packet error probabilities estimated by this protocol would have been unacceptable for use in our congestion control algorithm. As part of the SURAP 4 suite of protocols, we have designed a new parameter selection algorithm, described in detail in Section 2.4, which uses information about uncorrectable errors to estimate link quality.

Each node uses its measure of transceiver load to compute its bits-per-second ration according to the asymmetric ration control law, where in this case *threshold* represents the specified maximum acceptable data bits per second rate, *target* represents the specified maximum acceptable number of transmissions per packet due to contention, and *load* represents the computed average number of transmissions per packet due to contention.

The purpose of the threshold is to keep the bits-per-second ration from growing without bound when only a small amount of traffic is passing through a node. One could select a value for the threshold for each node, based upon the transmission speeds, error correcting encoding schemes, link error characteristics, average packet sizes, and average spacing values of each of the links associated with the node. However, all of these properties can change over time, and hence the threshold should be adjusted to reflect these changes. To eliminate the computational complexity associated with maintaining an adaptive threshold value, we opted for selecting a universal value that can be applied to all nodes at all times and that serves as a reasonable limiting value without constraining throughput. The value chosen for the simulations was 60,000 bits per second, which accounts for the maximum data packet size of 3000 bits and the maximum data packet rate governed by the moment of silence and the spacing delays imposed by the link-layer data transmission protocol.

The value of the target reflects the maximum average number of transmissions caused by contention that a node can tolerate without experiencing throughput degradation. As we have mentioned before, different traffic pattern and rate combinations result in different behaviors of the node, because of the nature of the contention process. Thus, the average number of attempts per packet transmitted that is sustainable without problems can also vary. For implementation simplicity, we settled on a single universal target value.

Given that contention exists, the average number of transmissions per packet under moderate load is greater than 1. Thus, it is necessary to choose a target value larger than 1 to avoid limiting throughput unnecessarily. However, the target should not be so high that it fails to detect imminent congestion. According to throughput studies of the SURAP 4 link-layer protocols [7], target values close to 2 result in an unacceptable number of data packet retransmissions. In the graphs depicted in [7], the average number of transmissions per packet is observable as the ratio of offered load versus carried load and appears relatively insensitive to packet size. We ultimately settled on a target value of 1.5, which appears to yield acceptable ration control in a variety of situations simulated.

3.3.2 Processor Load

At each node, the amount of processing time per data packet is on the order of 10 milliseconds of CPU time in the main protocol processor (Intel 8086). The processing time includes link-layer functions such as duplicate detection and acknowledgement generation and network-layer functions such as route determination. Thus, we expected CPU to be a critical resource to monitor for congestion control in the SURAN environment, given the substantial packet processing delay and the maximum transmission rate of 400,000 bits per second. In particular, in the presence of short packets from several neighbors, we expected the main processor to become the flow-limiting resource, even at the 100,000 bits per second transmission rate.

If the main CPU becomes overloaded, packets will accumulate in the buffers resulting in congestion. Hence, the node should anticipate congestion by monitoring the level of processor use. To measure CPU load, the node can use a low priority process whose sole purpose is to execute when there is no other work to be done. By monitoring the amount of time that it spends executing this low priority process during a specified time interval, the node can determine its idle time and hence its busy time. A node computes processor load as the percentage of busy time to total measurement time.

Each node uses processor load to compute a packets per second ration according to the asymmetric ration control law, where in this case *threshold* represents the specified maximum acceptable data packets per second rate, *target* represents the specified maximum acceptable percentage of CPU busy time, and *load* represents the computed average amount of CPU busy time during the past measurement interval. For our simulations, we selected a threshold value of 30 packets per second based upon maximum transmission speed, maximum packet size, and minimum link-level interpacket delays imposed by the moment of silence and spacing, and we selected a target processor load level of 90% based upon our experience with a similar congestion control algorithm implemented in wire-based packet-switched networks [1].

Surprisingly, the simulation results, discussed in detail in Section 4.4.3, show that under heavy load, a node's processor seldom becomes the limiting resource prior to any other. The inter-packet delays imposed by the link-layer protocols through the moment of silence and spacing are usually the primary factors restricting the packet rate through a node, in the presence of a small number of neighbors. The transceiver is normally the first resource to become saturated under load, regardless of packet size, in the presence of many neighbors. Nevertheless, there do exist cases where the packets-per-second rations based on processor load control flow rates during transient traffic bursts. Moreover, future protocols and hardware may change the SURAN environment such that processing capacity is more commonly a limiting resource. For these two reasons, we recommend implementing packets-per-second ration control as described above.

3.3.3 Output Link Throughput

Link quality refers to the probability of correctly receiving a transmitted packet, given that the transceiver is free of contention when the packet preamble arrives. Adjacent packet radio transmissions, jamming signals, ambient noise and distortions, distance, and receiver noise all contribute to poor link quality. As we have mentioned before, we have designed for SURAP 4 a parameter selection algorithm that provides an accurate estimate of the quality of a link. This estimate of link quality can be used as a control signal to boost power, signal energy, and FEC coding gain in order to improve link quality. However, it is not always possible to improve the quality of a link. If a link's quality falls below a given threshold, the link up/down protocol removes the link from service.

The quality of the link between a transmitting and receiving node pair coupled with the inter-packet delays imposed by the link-layer data transmission protocol determine the maximum rate of information flow between the two nodes. If the flow rate into a link exceeds the flow rate out of a link, packets will queue behind the output link, resulting in congestion. Flow rate mismatches are more likely to occur at poor quality links where retransmissions are necessary to promote successful packet forwarding to the neighbor. We expected that in such cases, the effective throughput of the output link could become the resource limiting packet flow to the neighbor.

To determine the effective throughput of one of its output links [illegible], an node must compute the expected number of transmissions per packet due to link errors only, which requires knowledge of the expected packet error rate. The node must keep track of the average packet size during each measurement interval so that it can compute the expected packet error rate, [illegible], from the bit error rate supplied by the parameter selection algorithm.

The expected number $E[N\ ]$ of transmissions per packet on link j, resulting only from link errors is equal to

$$
\begin{aligned}
E[N\ ] &= (1 \quad)(\sum_{=}^{6} \quad + \sum_{=7} 6 \quad) \\
&= \frac{(1 \quad)}{1} \quad ^{6} \\
&= \sum_{i=}^{6} \quad .
\end{aligned}
$$

The delay per packet on link j – D – is equal to

$$D_j = L\ E[N_j] = L_j \sum_{i=1}^{6} r_j^{\ -1}.$$

where L is the transmission delay of an average size packet plus the minimum spacing delay on link j. Therefore, the effective throughput of link in packets per second is —

Each node uses output link throughput to compute each of its links packets-per-second rations according to the asymmetric ration control law, where in this case *threshold* represents the specified maximum data packets per second rate, *target* represents the maximum effective throughput of the output link computed during the past measurement interval, and *load* represents the number of distinct packets offered to, but not necessarily accepted by, the output link during the past measurement interval. Although we could have specified a threshold value dependent upon characteristics specific to the given link, we opted for a universal value of *threshold* that can be applied to all links, in order to simplify implementation. The value selected is 30 packets per second, the same as that for processor-driven ration control.

As nodes are not permitted to keep in service links with packet error rates greater than 10%, retransmissions caused by channel noise do not seem to be a limiting factor for output link throughput. However, the parameter selection algorithm can potentially introduce an eight-fold packet overhead on a link, at lowest FEC rate and slowest channel bit rate, which radically reduces output link throughput especially in the presence of medium to large size packets. In this case, control of rations according to output link throughput appears necessary to avert congestion under load. We present simulation results that corroborate this hypothesis in Section 4.4.

3.3.4 Buffer Use

In the current LPR, there are only 11 buffers available for holding data packets and their acknowledgements. To maximize throughput, the node should permit buffer sharing among competing traffic flows. To reduce the chance of lockout, the node should limit the number of buffers that can be consumed by the aggregate flows from any one neighbor. However, these are conflicting goals. In our simulations, we have chosen to resolve the conflict as follows. Each node has nine buffers to be shared among traffic destined for all output links and two buffers to be reserved for accepting traffic from hosts and from input links. Congestion control is responsible for promoting fair use of the limited buffer supply, as we describe below.

We initially rejected buffers as a resource to monitor for congestion control for the same reason that we have avoided other delay-based performance measures: extensive buffering is a symptom and not a predictor of congestion. However, after we discovered through simulation that processor load and output link offered load were not universally effective indicators of imminent congestion, we began to experiment with buffers. In the presence of a small number of neighbors, transceiver contention need not be an issue: it is the inter-packet delay imposed by the link-layer data transmission protocol that has the most significant impact on the rate of packet flow through the node. As the load becomes heavier, packets queue waiting for service at the output links, and so buffer use is a direct measure of congestion.

The node uses average buffer occupancy to control packets per second rations. We had previously considered buffer-oriented congestion control using information about packets rejected (NACKed) because of insufficient buffer space. However, control in response to NACKs is reaction to rather than anticipation of congestion.

Each node computes average buffer occupancy during a given measurement interval by sampling the number of buffers in use each time it receives a distinct (non-duplicate) packet. The node maintains a running sum of the buffer occupancy samples and a running sum of the number of distinct packets received, which it uses to compute average buffer occupancy during a given measurement interval. The measurement interval for buffer use is an integer multiple of the measurement interval used for transceiver (and processor and output link) load. The reason is that the variance of buffer occupancy is expected to be higher than that of transmissions per packet: hence a longer measurement interval helps to obtain a more accurate measure of average buffer use. However, as we explain in Section 4.6, buffer use computed in this way still does not eliminate steady-state oscillation of ration values at the buffer bottleneck. We are currently experimenting with using a time average of buffer load to smooth out oscillations. However, the simulations presented in this report were all run with the buffer occupancy described above.

Each node uses buffer occupancy to compute a packets-per-second ration according to the asymmetric ration control law, where in this case *threshold* represents the specified maximum acceptable data packet per second rate – 30 packets per second, *target* represents the specified maximum acceptable buffer occupancy, and *load* represents the computed average buffer occupancy during the past measurement interval. For our simulations, we selected a target value of 6 buffers, based upon the goal of averting congestion before it occurs and upon the observed mean and variance of buffer occupancy in the presence of transceiver-based congestion control.

3.4 RATION DISTRIBUTION

Each source node depends upon the ration distribution mechanism to provide it with timely and accurate ration information for those resources situated along each of its paths to its destinations. The node uses this information to determine the bits-per-second and packets-per-second flow rations for each path, each of which is equal to the minimum of the corresponding resource rations on the path. We considered three separate methods for ration distribution, one of which was selected for the congestion control algorithm.

3.4.1 Flooding

The first method is a flooding scheme in which each node periodically distributes its bits-per-second and packets-per-second ration information to all other nodes in the network. Although simple, this method wastes network resources, since seldom do all nodes require ration information from all other nodes.

3.4.2 Tagging

The second method is referred to as "tagging." Each source node periodically requests ration information from the nodes along each of its paths, by tagging data packets. Each tagged packet contains two special header fields, one for bits-per-second ration information and one for packets-per-second ration information. Before the source node sends the packet along the path, it places its bits-per-second and packets-per-second rations in the special header fields. When a node receives a tagged packet, it compares its own bits and packets per second rations to those carried in the packet and overwrites a packet value with its own value whenever its own value is smaller.

When a tagged packet reaches its destination, it contains the bits-per-second and packets-per-second flow rations for the given path. The destination node returns these values to the source node in special fields in the data packet header, similar to those used in collecting the rations. Note that a tagged packet may simultaneously collect rations in one direction while returning rations collected in the opposite direction. Tagging is an improvement over flooding since it consumes network resources only along those paths where ration information is requested. Furthermore, rations may be collected more frequently with tagging than with flooding, since tagging is less of a resource drain, and thus congestion control can be more responsive. However, tagging does have a disadvantage: when a resource becomes saturated, tagged packets may queue behind the resource, retarding the distribution of ration information when it is most needed.

3.4.3 Backpropagation

Backpropagation, the selected method, is similar to tagging in that rations are collected only from those resources along the paths selected by the source node. The source node A issues a request, in a data packet, for flow ration information along a given path to destination Z. The first node B on the path responds to the request by returning to the source, in the link-level acknowledgement, the bits and packets per second flow rations from itself to Z and by propagating the request in the data packet along to the next node C on the path. A uses the flow rations returned in the acknowledgement from B to compute its flow rations to Z by taking the minimum of its own resource rations and B's flow rations to Z. When using alternate routing, A computes its flow rations as the minimum of its resource rations and the flow rations in all acknowledgements received for alternate-routed copies of the original packet, in order to avoid saturating any of its alternate routes.

Backpropagation continues as C responds to B's flow ration request by returning, in its acknowledgement to B, the flow rations from itself to Z and by propagating the request in the data packet to the next node on the path. B uses C's returned flow rations to compute its own flow rations to Z. The process terminates when the requesting data packet reaches the destination Z.

Backpropagation relies on the hereditary property of routing. Path heredity is illustrated as follows. If a node A's path to node Z passes through an intermediate node M, then the portion of A's path from M to Z is identical to M's path to Z. Moreover, the minimum function has a similar property, namely $\min\{x_1,\ldots,x_n\} = \min\{\{x_1,\ldots,x_j\}, \min\{x_{j+1},\ldots,x_n\}\}$, for any $1 \leq j \leq n$. Together these two properties imply that if the first intermediate node on A's path to Z is B, then the bits or packets per second flow ration from A to Z is the minimum of X's corresponding bits or packets per second resource ration and the corresponding flow ration for the path from B to Z.

With backpropagation, the delay experienced by a source node in obtaining ration information from the bottleneck resource along a given path is influenced by the number of path hops between the source and the bottleneck, the number of flows that use a path containing the section between the source and the bottleneck, and the frequency of requesting ration information. Note, however, that flow ration information is not delayed by congestion along the path, since it is obtained directly from a neighbor.

3.5 RATION ENFORCEMENT

Each source node enforces the flow rations to each destination node with which it communicates, using timer-based mechanisms referred to as *throttlers*; there is one throttler per communicating destination. Each throttler determines whether or not a packet from a local host at the source may be submitted to the given destination. If the throttler determines that a packet may be submitted to the destination, it computes the amount of time that must pass before the next packet may be submitted to that destination. This time interval is the minimum of the reciprocal of the packets-per-second ration and the product of the reciprocal of the bits-per-second ration and the packet's length in bits. Until this time interval expires, the throttler will not submit any other packets to the given destination; packets accepted from a local host for the given destination must wait in the node during this time interval.

4. PERFORMANCE RESULTS

4.1 INTRODUCTION

In this section, we discuss the performance of tr algorithm as it relates to fairness, rate of convergence, and robustness. We use simulation results and simple models to emphasize the important points.

For the discussion on fairness, we describe in more detail the nature of resource bottlenecks. The main factor in determining which resource becomes the traffic bottleneck in the network is packet size. For large packets, the transceiver is the bottleneck resource since getting traffic into nodes is constrained by contention. If traffic consists of small packets, the transceiver can sustain a higher throughput rate. The service rate of the node typically determines the limit of the throughput rate of the node in this situation. Characterization of the service rate of a node is a difficult problem, since it depends on the contention experienced by the different output links and on the traffic pattern flowing through the node.

Monitoring buffer use is one way to detect service rate problems. The cross-over point between transceiver bottleneck situations and buffer bottleneck situations is also dependent on the traffic pattern. For packet sizes corresponding to this operating point both resources will participate in flow control, with the burden switching randomly from one to the other depending on local conditions.

The CPU as a bottleneck requires a situation with several flows from different neighbors crossing at a node. Typically, even with small packets, high contention follows and the transceiver becomes the bottleneck. However the CPU rations play a positive role during transients for cross traffic, as mentioned above, given the parameters that apply in SURAP 4. We have recently observed that a combination of a few flows crossing at a node and small enough packets (500 bits) has sometimes led to a sustained interval (tens of seconds) over which CPU rations regulate the traffic flow, before transceiver rations or buffer rations take over control; however, we have not included graphs of these simulation results in this report.

Output link throughput is observed to become a bottleneck primarily when a link requires large packet overhead; specifically, when a link requires maximum link gain to maintain communication. If all other flows to and from a node use input and output links operating at 400 kilobits per second and no FEC, while one output link requires 100 kilobits per second and rate 1/2 FEC, the throughput for this output link may be sufficiently lower than for the rest: if its traffic is not throttled the unrestricted backlog produced in the node will lead to buffer throttling of all flows. This action is unfair to flows not using the low throughput output link.

Oscillation of ration values in steady state is observed in many of the simulation results presented. The nature of these oscillations was discussed in Section 3 and will be treated again in Section 4.6.

In Section 4.2, we explain the procedure followed in the simulation experiments. In Section 4.3, we compare throughput and performance of a sample topology with and without the congestion control algorithm. In Section 4.4, we discuss models for resource bottlenecks. These models are used in Section 4.5 to explain max-min fairness operation. In Section 4.6, we discuss the behavior of the rate of convergence as the size of the network increases. In Section 4.7, we evaluate the algorithm's robustness in the face of node and link failures. The exact value of the network parameters used for simulation are listed in Appendix A.

A large number of parameters must be set in the congestion control algorithm. We have run simulations expected to be representative of a large part of the space of network operation and set parameters accordingly. An exhaustive test is not practical, but we expect that as we gain experience in more network topologies and situations, and as the algorithm gets tested in more detailed simulations, some of the parameters will be altered accordingly, and the design may be fine tuned. However, we think that the present report gives a solid review of the performance expected of the algorithm.

4.2 SIMULATOR

We developed an event-driven software simulator to test the performance of the congestion control algorithm. The software is written in Common-Lisp and runs on a Symbolics 3670 Lisp Machine.

The software simulates the SURAP 4 architecture, including all the protocols described in Section 2, with the exception of alternate routing, parameter selection, and link up/down algorithms. The simulator uses static minimum-hop routing to compute paths there are no routing updates – PROPs – broadcast.

For the topologies simulated in the discussions to follow, simulation of alternate routing is not relevant. This does not mean that alternate routing has no impact on the performance of our congestion control algorithm; it means that, for the topologies discussed, no alternate routing paths exit for the traffic situations simulated.

Regarding parameter selection, links have the software option of 100Kbits per second versus 400Kbits per second, the option of several FEC coding rates, and the option to set bit error probability on the link. These parameters remain constant during a simulation, instead of being adaptive as they are for the parameter selection algorithm. The bit error probability parameter is intended to summarize the effect of transmission power, distance between radios, and FEC rates. Because the parameter selection algorithm is designed to maintain a packet error rate, due to noise, of less than 0.1, we have set the bit error probability equal to zero in all our simulations. Except for two sets of simulations, all links in the network have been set to run at 100Kbits per second and no FEC coding. The first exception is the set of simulations in Section 4.4.3 for CPU bottlenecks. In these, links were set to 400Kbits per second channel bit rate and no FEC. The intention was to have as little channel contention as possible. The second exception is the set of simulations in Section 4.4.4 for output link throughput. In these, one output link was set to operate at 100Kbits per second, 1/2 FEC rate, while all other links were set to 400Kbits per second, no FEC. The intention was to provide a link with significantly higher service time than the rest.

There exist two other discrepancies between the link-layer protocols as designed for SURAP 4 and as implemented in the simulator, both relating to retransmissions. The retransmission interval to which linear backoff is applied is designed to be much longer than the spacing interval in SURAP 4 and to be close to the spacing interval in the simulator implementation. The K-buffering backoff strategy is based on a time average of packet service time in SURAP 4 and on the most recent sample of packet service time in the simulator. We do not expect these discrepancies to cause fundamentally different behavior of the congestion control algorithm in the two environments.

Traffic generation is simulated as a Poisson process. The mean value for the exponential interarrival time distribution is supplied by the user. Packet lengths are generated through a truncated exponential distribution whose minimum, maximum and mean values are also user-supplied. The distributi n is truncated by assigning the probability mass below the minimum value to the minimum value itself, and the probability mass above the maximum value to the maximum value itself. The capabilities of the software are more general than described here; we mention here only the options that apply to the simulations discussed.

Processing times at a node are simulated by deterministic waits whose values are considered representative of the hardware in the LPRs. Description of this hardware can be found in [2]. In congestion control simulations, source hosts submit traffic at a rate of 20 packets per second to their attached nodes and of mean length specified by the user. When a host flow is throttled by the congestion control algorithm, packets are buffered at the host until they can gain entrance to the network. The value of 20 packets per second was chosen to provide network saturation in the absence of congestion control whenever two or more flows converge. Exceptions to this implementation are noted as they occur. Mean packet lengths chosen are detailed in the relevant sections. In the SURAP 4 architecture, packet lengths can vary from a few tens of bits to over 3000 bits. We will often attempt characterizations of a network under large-packet traffic and under small-packet traffic. For most simulations presented, large-packet traffic consists of 2500 bits mean packet length and small-packet traffic consists of 500 bits mean packet length.

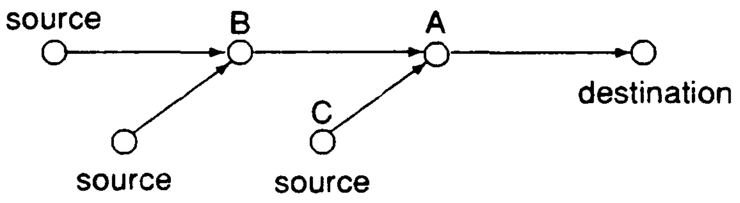


Figure 4.1: NETWORK FOR THROUGHPUT AND DELAY TESTS

We often use the steady-state time average path ration values as the steady-state source flow in the networks being simulated. This identity is justified by the fact that source nodes, in the algorithm design and in the network simulations, enforce path ration values. Thus, if the simulated network achieves steady-state feasible flows, the ration values and the traffic flows must be equal in mean value.

Appendix A lists the network parameters used in the simulations. Unless otherwise stated in particular sections, the reader should assume that these are the parameters that apply to the simulation being discussed. Exact knowledge of their value will not be necessary for the discussions.

More recently, we have been experimenting with a packet radio simulator implemented using the Opnet™ simulation package. This simulator emulates all of the SURAP 4 protocols as they would be implemented in the LPRs. The results obtained from the two simulators are remarkably similar, indicating that the simplifying assumptions used in developing the Lisp-based simulator do not mask the basic algorithm behavior characteristics. In the future, we intend to make almost exclusive use of the Opnet™ simulator, because of the greater detail available and the cleaner user interface.

4.3 THROUGHPUT AND DELAY

Throughput and delay performance evaluation are based on a comparison between simulation experiments run with and without congestion control for a chosen topology. The goal is to illustrate that the Congestion Control Algorithm does not limit throughput severely and is effective in providing congestion control, as evidenced by the delay characteristics obtained.

The topology chosen appears in figure 4.1.

Figures 4.2 to 4.9 depict the results of simulations run on this network with and without congestion control. For large packet traffic, the network throughput is slightly reduced by the congestion control algorithm with respect to the uncontrolled situation. Delay however is greatly improved. The transient regime for the situation using congestion control is approximately 40 seconds long, which can be observed from the plot of average delay. The packet length distribution consisted of a truncated exponenential distribution with mean of 2500 bits, minimum packet length of 224 bits, and maximum packet length of 3000 bits. For small packet traffic, the network throughput actually improves, given the absence of heavily backlogged nodes. Delay is also greately improved. The transient regime for the situation using congestion control in this case is approximately 20 to 30 seconds long.

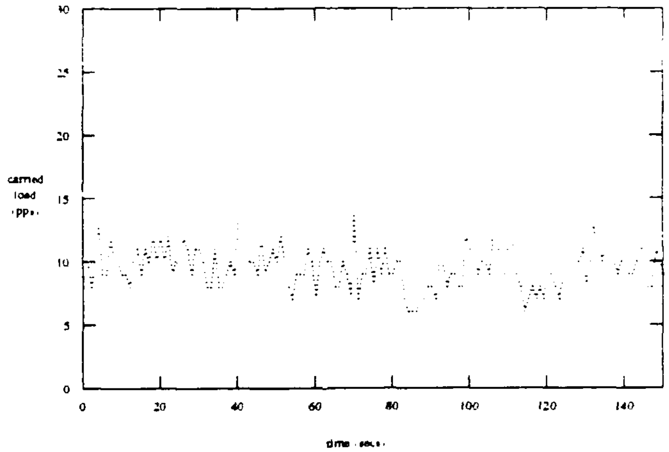


Figure 4.2: CARRIED LOAD WITH CONGESTION CONTROL (LARGE PACKETS)

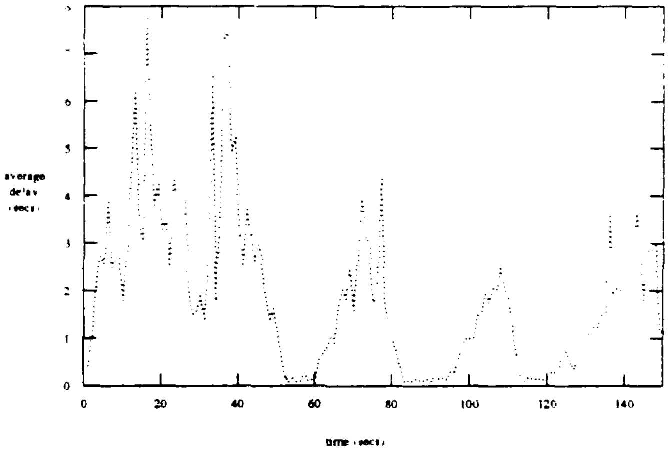


Figure 4.3: AVERAGE DELAY WITH CONGESTION CONTROL (LARGE PACKETS)

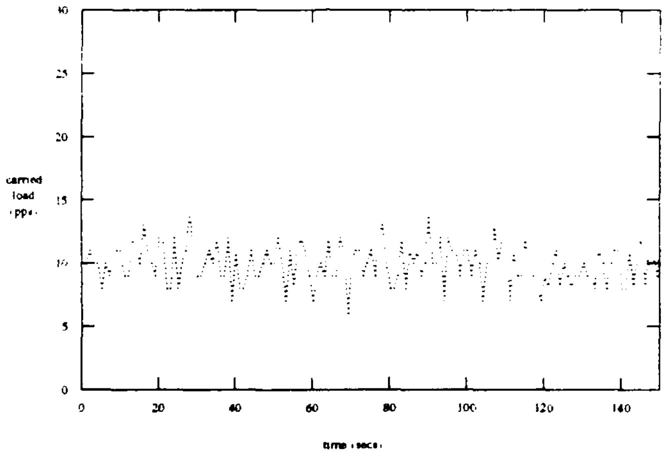


Figure 4.4: CARRIED LOAD WITHOUT CONGESTION CONTROL (LARGE PACKETS)

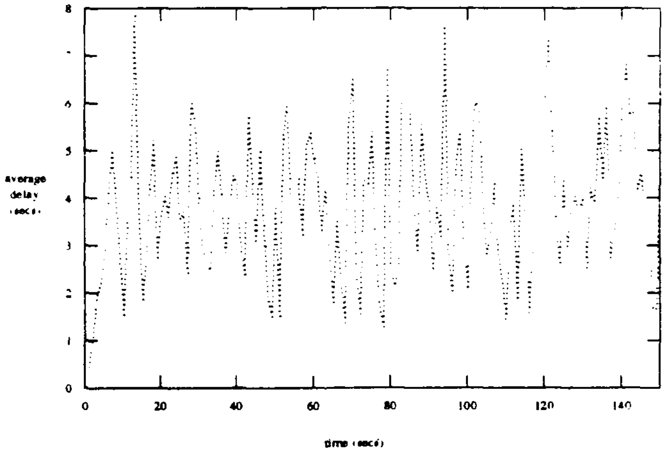


Figure 4.5: AVERAGE DELAY WITHOUT CONGESTION CONTROL (LARGE PACKETS)

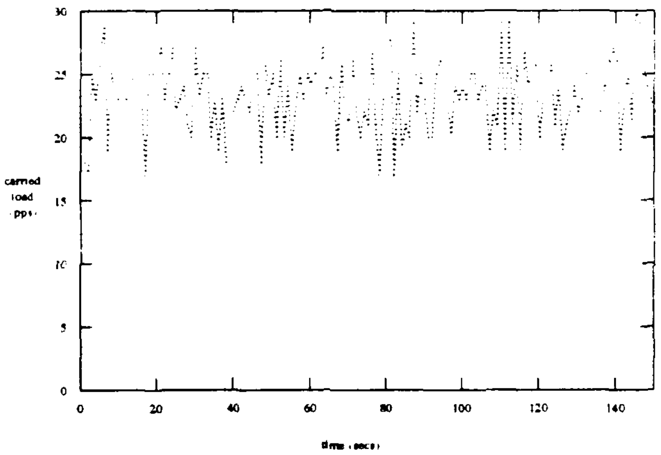


Figure 4.6: CARRIED LOAD WITH CONGESTION CONTROL (SMALL PACKETS)

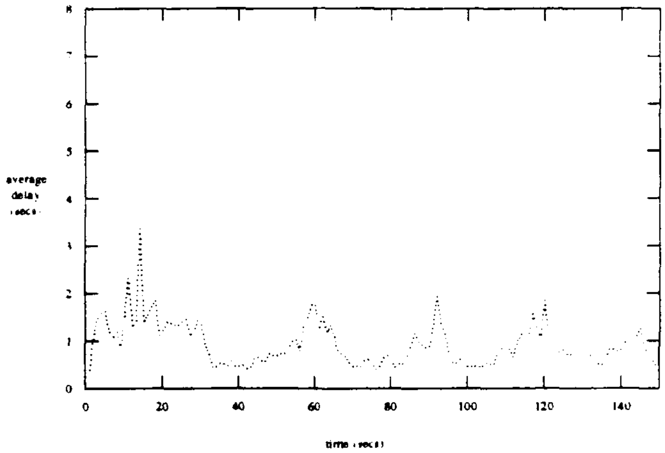


Figure 4.7: AVERAGE DELAY WITH CONGESTION CONTROL (SMALL PACKETS)

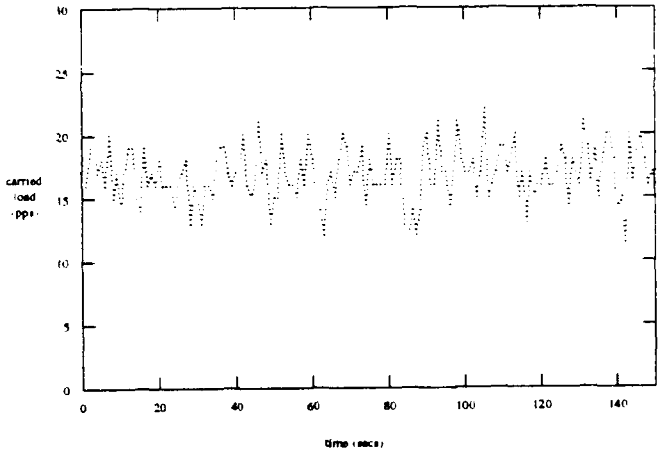


Figure 4.8: CARRIED LOAD WITHOUT CONGESTION CONTROL (SMALL PACKETS)

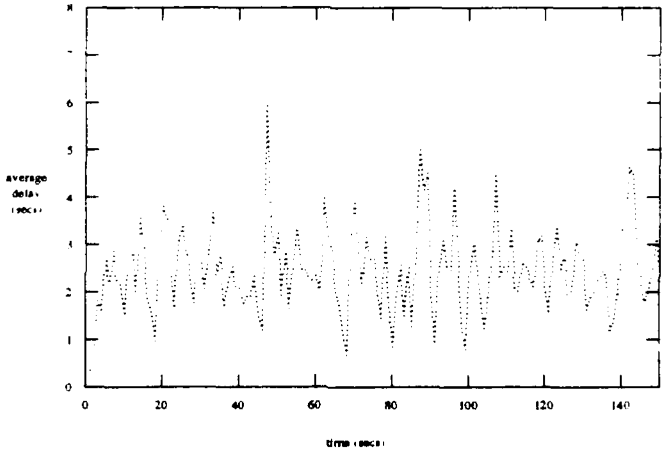


Figure 4.9: AVERAGE DELAY WITHOUT CONGESTION CONTROL (SMALL PACKETS)

4.4 BOTTLENECKS

We use simplified mathematical models of the network dynamics as a means of facilitating the discussion of the performance of the congestion control algorithm. In the simplifications, some accuracy and realism is lost. However, some of the most fundamental aspects of the dynamic behavior of the network are captured and illustrated by the following models.

4.4.1 Contention Model

For traffic with large packet lengths, transceiver bottlenecks are likely to become the primary cause for congestion. Since the transceiver utilization metric is based on the average, over all input links, of the number of attempts per packet sent to the node, we are interested in the dependence of this quantity on traffic flow. For an arbitrary packet sent to a node A, the average number of attempts is a monotonically increasing function of the amount of traffic it has to contend with at the transceiver of this node. The amount of traffic, in turn, is assumed proportional to the duration of packets on the channel and to the traffic rate, in packets per second, handled by this transceiver. The contention process can be divided into two components:

1. Reception: transceiver is receiving a packet from some other neighbor.
2. Transmission: transceiver is transmitting a packet to one of its neighbors.

In this model, we assume perfect capture so that once the transceiver synchronizes to a packet the packet is received correctly even if other packet transmissions interfere with it before it has been completely received. This simplification is unnecessary since the effect of interfering transmissions resulting in packet reception errors is similar to the effect of traffic that contends for the same transceiver.

The reception component of contention must exclude traffic from the node sending the test packet under discussion, while the transmission component can include traffic from node A to the node sending the test packet. For the purpose of notation, let λ_j denote the carried load, in packets per unit time, from a neighbor j sending to A and λ'_k be the carried load from A to a neighbor k. Carried load means the steady state, sustainable packet throughput rate. The reception component of the contention process refers to the carried load from other neighbors into node A since this load consists of packets that successfully "capture" the transceiver. Instead, the second component of contention refers to the offered load from A to any of its neighbors, that is, carried load plus retransmissions. In steady state, and without congestion, the offered load equals the carried load to a node k, times the average number of transmissions per packet on that link, r'_k. Host traffic generated at the node shows up as carried load in the transmission component.

For simplicity, we first consider situations with constant packet duration t for all packet transmissions in the network. The effect of dropping this assumption is considered after the main issues have been discussed. The average number of transmissions per packet into A, $< r >$, is thus monotonically increasing in

$$t\left(\sum_{\phi} \lambda_j + \sum_{\mathcal{N}} \lambda'_k r'_k\right) \tag{4.1}$$

where ϕ denotes the set of nodes transmitting to node A and $\mathcal{N}$ denotes the set of nodes that A transmits to. The first and second terms within parentheses correspond to the reception and transmission components of contention above, respectively. When packet duration is not constant for all transmissions, we still expect contention to be proportional to the average packet duration on the links. For homogeneous traffic over network links and all links with equal channel bit rate and FEC rate, the average duration would be equal for all links. However, even with constant packet lengths, the parameter selection algorithm may still render the packet duration different on each link and a term of the form $\sum_{\phi} \lambda_j t_j + \sum_{\mathcal{N}} \lambda'_k r'_k t'_k$ is in order. This discussion could be generalized to that case. The simplification is adopted for simplicity of treatment.

One aspect ignored in this model is the fact that, for a given amount of traffic into node A, the contention experienced increases with the number of neighbors sending to A, that is, the probability of a packet finding the transceiver busy at A is expected to increase with the number of neighbors sending to A, given a fixed aggregate carried load to A. This is because of the greater asynchronicity among transmitting neighbors. Similarly, if most traffic into A comes from one neighbor, that stream experiences lower contention than all the other links, each of which individually contends with the large volume of traffic from this neighbor. Since $< r >$ is computed over the set of neighbors transmitting to A, this latter situation is likely to lead to a higher value of $< r >$ than if the traffic is uniformly distributed among these links. In conclusion, the number of input links and the distribution of traffic among them also has an impact on $< r >$ which is not accounted for in (4.1). The effect of these two factors is to modify the reception component of contention. Evenly distributed traffic, $\sum_{\Phi_1} \lambda_j$, from a small number of neighbors (close to two) is expected to induce a value of $< r >$ equivalent to a situation with evenly distributed traffic, $a \sum_{\Phi_2} \lambda_j$, from an infinite number of neighbors, with $0 < a < 1$ reflecting the synchronicity effect. Non-uniform traffic, $\sum_{\Phi_1} \lambda_j$, is expected to induce a value of $< r >$ equivalent to a situation with evenly distributed traffic, $b \sum_{\Phi_2} \lambda_j$, with $b > 1$ reflecting the higher contention experienced by smaller traffic links in the non-uniform situation.

4.4.2 Queuing Model

The number of buffers in use at node A is modeled as a queuing process. The arrival rate is the steady state carried load into the node and the service rate is the rate at which the node can dispose of the packets. We consider the buffers in use due to traffic through a relay node A. Packets at A intended for a neighbor k are said to be queued for the "output link" k, referring to the directed link from A to k. For simplicity we model this queue as an M/G/1 queue. Important assumptions in the queuing model are

1. Independent Poisson Arrival processes for the carried load from each neighbor transmitting to node A. The service time for an output link is independent of the arrival instants of packets intended for that link.
2. All packets queued at A for the same output link see independent, identically distributed service times.
3. In the absence of cross-traffic, all (output) links in the network have identical service-time distributions as a function of the number of retransmissions on the link.
4. No host traffic is generated at A.
5. The queue at A is infinite.

Let us look more closely at the weaknesses in the model implied by these assumptions. The independence among arrival processes in assumption 1 ignores synchronization effects among packet radios. The randomization inherent in the retransmission strategy, and the randomness introduced by successive forwarding nodes, contribute to breaking up synchronization among nodes. However, due to the broadcast nature of the channel, some synchronization is expected since nearby nodes often observe the same events in the common channel. Independence between arrivals times and service times implies that arrival instants are independent of packet lengths and that link scheduling delays are independent of packet arrival times. Independence between arrival instants and packet lengths is reasonable as long as packet duration is only a fraction of typical packet interarrival times. Otherwise the packet length will be determinant in the packet interarrival times. This argument also assumes no synchronization effects among nearby nodes. Independence between link scheduling delays and packet arrival instants ignores the fact that, upon arrival of a packet, traffic being handled by the node processor, and scheduled for the transceiver, depends on which link this packet arrived through. The traffic present is likely to consist of packets from other links and hence their service processes will reflect their own flow characteristics. Under homogeneous network traffic, without synchronization effects, the assumption is more reasonable.

Assumption 2 violates the fact that the packet lengths affect the service time. However, the assumption is equivalent to assuming that once a packet is in the node, it is scheduled independently of its length, and that just before it is transmitted the length of the packet is obtained out of a probability distribution reflecting source packet lengths. With homogeneous traffic length distributions the assumption is appropriate for the model.

Assumption 3 ignores the fact that the parameter selection algorithm affects the packet transmission duration and hence the average service time. Cross-traffic increases a link's service time since the same transceiver and node processor must be shared by all output links. For simplicity we initially treat a situation without cross-traffic and with equal parameter selections for all links, and consider the effects of dropping these assumptions later.

Assumption 4 is also for simplicity, since rules for buffering of packets from a host differ from those applied to packets from neighbors. The effect of dropping this assumption is discussed later on.

Assumption 5 is partially justified by the fact that the target value recommended for buffer-based ration control is close to one half of the total number of buffers available and the value of K recommended for K-threading limit is close to the total number of buffers. However, during transients the infinite queue assumption is not a good approximation.

We feel that most of these assumptions are not critical to justify the results obtained. They are introduced to simplify the model and its discussion, so that the important issues involved can be brought out without swamping them in details.

Let λ_j be the carried load from node j to node A and λ'_k be the carried load from node A to k. Because of K-Buffering, at most K packets from neighbor j can be stored at the node. The mean service time for an output link k is a monotonically increasing function of the number of transmissions per packet and is denoted by $s_k(r'_k)$. The link load factor ρ_k is defined as $\rho_k = \lambda'_k s_k(r'_k)$. We model the number of buffers in use due to traffic flowing to a given neighbor k as an M/G/1 queue. We omit the subscript k from the queue parameters to simplify notation. The output link that the queue corresponds to should become apparent from the context. Let $\overline{n}$ be the steady state number of buffers in use at node A scheduled for link k. The total, steady state, number of buffers in use at node A for output link k is a function of the total traffic through A for link k. For the M/G/1 model, we have

$$\overline{n} = \rho + \rho^2 \frac{(1 + C^2)}{2(1 - \rho)} \tag{4.2}$$

where C is the coefficient of variation for the output link. The fraction of buffers due to neighbor j sending through A is $\overline{n}(j) = \left(\lambda_j \;\; \sum_{\phi} \lambda_j\right) \overline{n}$. Since the service time is assumed independent of the arrival process, the coefficient of variation is just a parameter of the output link and (4.2) is a monotonically increasing function of ρ. This observation allows us to determine the resource bottleneck due to buffers in situations similar to the ones considered for transceiver bottlenecks, and to discuss the implementation of max-min fair flows.

4.4.3 CPU

The existence of link flow control and channel access protocols tends to preclude the possibility that traffic from a single neighbor will lead to CPU service problems. In our case, the spacing algorithm allows time for CPU tasks to complete when determining interpacket waits. When a node is the hub for cross traffic, interleaving packet processing of different streams is possible for the node. Depending on hardware and link protocols, this situation may lead to a CPU bottleneck. We have specifically simulated a situation consisting of a node with 16 neighbors, the maximum permissible in SURAP 4, each forwarding traffic to a different destination through this node. Small average packet length traffic was chosen in an attempt to keep the transceiver from driving the congestion control process. Minimum, maximum, and mean packet lengths were 60 bits, 3000 bits, and 100 bits, respectively. Hosts were attempting to submit traffic at a rate of 1.5 packets per second into the network.

We discuss both the transient behavior of the ration values and their steady state behavior. Only data for one of the 16 cross-traffic streams is discussed. This data is representative of the behavior of all the streams in the simulations.

Figures 4.10 and 4.11 show the bits-per-second and packets-per-second rations for a simulation experiment using *target* = 0.75 for the CPU. They illustrate that the CPU driven packets-per-second rations are more effective in providing flow control during the transient given the 100 bit mean-packet-length traffic simulated[1]. Both rations converge down from their maximum value. At 14 seconds the value of bit-per-second rations is approximately 700 bits per second, corresponding to about 7 packets per second: and at 15 seconds it becomes about 290 bits per second, or about 2.9 packets per second. Before second 15, the packets-per-second rations correspond to smaller packet flow than the bits-per-second rations. Between 15 and 20 seconds, the flows are comparable. Figures 4.12 and 4.13 show the steady state behavior of the two kinds of rations. The graph for bits-per-second rations appears discontinuous because the rations momentarily surpass the scale in the figure. We have preferred to only plot this range to allow easy comparison between bits-per-second rations and packets-per-second rations. Both rations oscillate in steady state and are approximately equivalent in terms of packets-per-second flow. Packets-per-second rations seem to experience a smaller amplitude of oscillation and overall seem to correspond to a smaller flow, effectively determining the flow control. The time average packets-per-second ration value, over the interval 20 to 150 seconds, is 2.28 packets per second. The time average bits-per-second ration value, over the same interval, is 264 bits per second. The higher flow peaks (or ration peaks) of bits-per-second rations compared to packet-per-second rations observable in the figures may be a result of the fact that the average number of attempts per packet decreases more rapidly as a function of flow decrease than the CPU load. In other words, these higher peaks may reflect the marked nonlinearity that exists for the number of packet transmissions as a function of total flow.

When these simulation experiments are run using *target*=0.9 for the CPU, the ration behavior shown in figures 4.14 to 4.17 is obtained. The transient behavior for packets-per-second rations is similar to that of the previous experiment. This is not very surprising for the following reason. Under a sudden burst of traffic resources are saturated, and the ratio between *target* and *load* remains fairly constant until rations converge to values close to feasible flow and the accumulated backlog disappears. Over steady state however, the bits-per-second rations are primarily responsible for flow control while the packets-per-second rations return to maximum (not shown in the figure). This figures confirm that packets-per-second rations in the cross-traffic experiments run are driven by CPU load, instead of buffer use or output link throughput.

[1] That the packets-per-second rations are driven by CPU load can be confirmed by running the same experiment with a CPU *target* = 0.9, as shown in figures 4.14 to 4.13

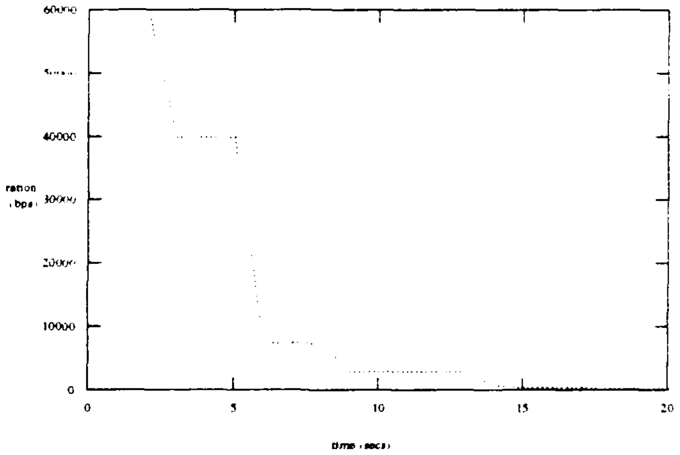


Figure 4.10: TRANSIENT BPS RATIONS FOR CPU TARGET = 0.75

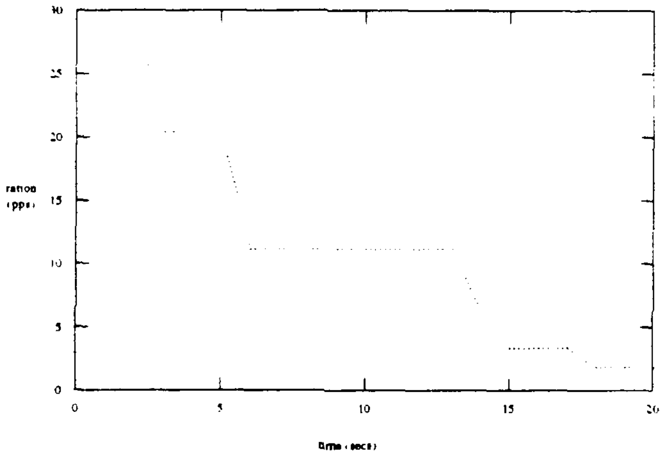


Figure 4.11: TRANSIENT PPS RATIONS FOR CPU TARGET = 0.75
The packet rations are more effective than the bit rations during this transient.

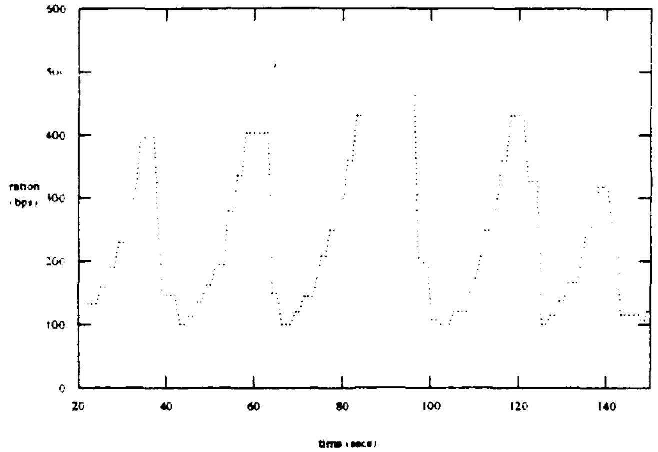


Figure 4.12: STEADY STATE BPS RATIONS FOR CPU TARGET = 0.75

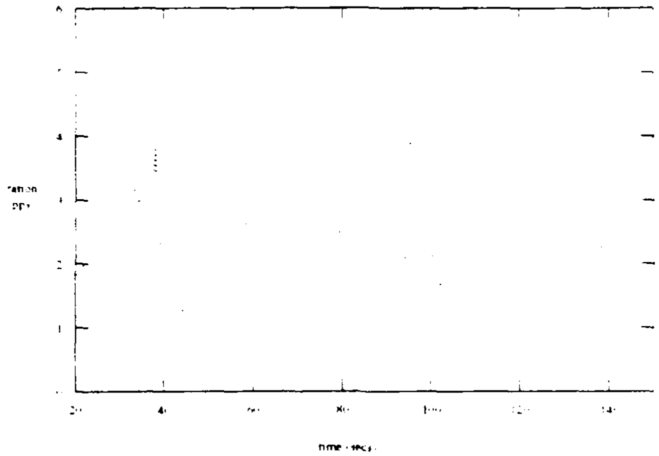


Figure 4.13: STEADY STATE PPS RATIONS FOR CPU TARGET = 0.75
The packet rations limit flow more than the bit rations for this simulation.

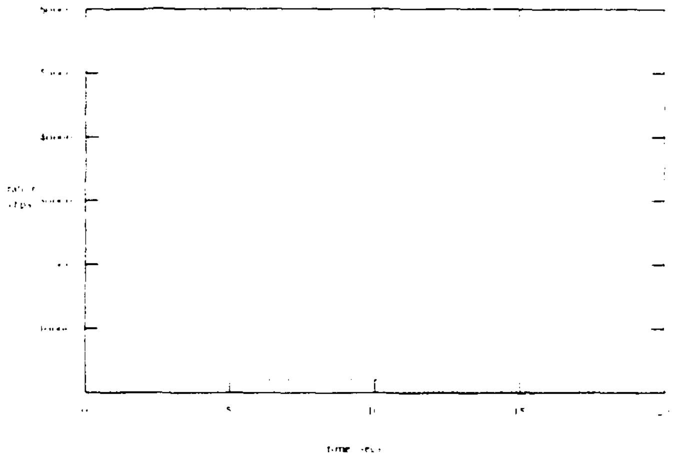


Figure 4.14: TRANSIENT BPS RATIONS FOR CPU TARGET = 0.9

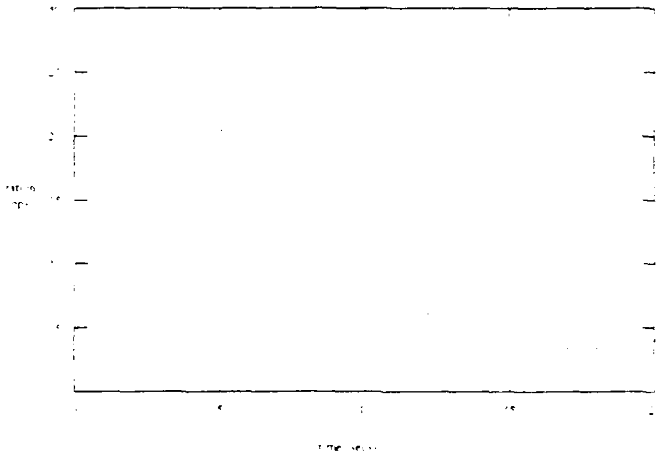


Figure 4.15: TRANSIENT PPS RATIONS FOR CPU TARGET = 0.9
The packet rations are more effective than the bit rations during this transient.

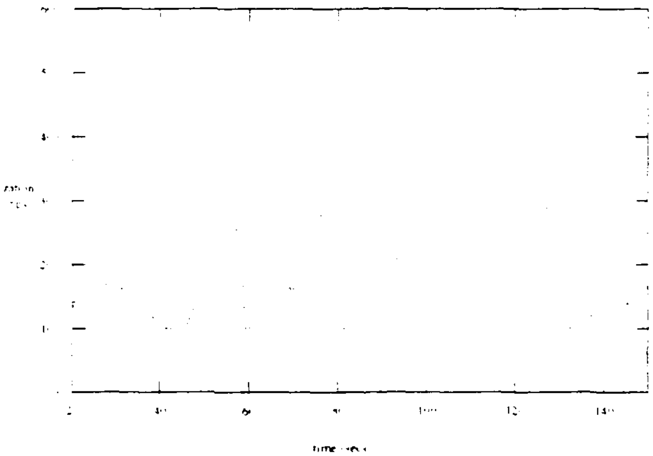


Figure 4.16: STEADY STATE BPS RATIONS FOR CPU TARGET = 0.9
The bit rations determine the flow in this simulation.

The appropriate value of target CPU load will depend on the choice of hardware and microcode running on the node processor. From experience with other types of computer networks we feel that utilizations close to one are permissible with most choices of hardware. This seems consistent with the asynchronous, flexible nature of CPU task scheduling. Since the CPU is a node server, load too close to one will lead to queuing problems. In the CPU experiments run, buffer backlog did not become a problem, as evidenced by the lack of buffer driven control. These observations, and the small packet durations chosen for the traffic makes us conclude that CPU is not a critical resource in SURAP 4. As mentioned earlier however, CPU rations seem to play a useful role in flow control during small-packet transient cross-traffic situations. In a sudden rush of small-packet traffic, the maximum bits-per-second ration is equivalent to many more packets than the maximum packets-per-second ration, and bits-per-second rations take longer to converge to feasible flow values. This role would be redundant if bits-per-second rations used measurements of packet duration to determine how fast to decrease, at the expense of more complexity in the computations and fine tuning required.

The experiments underline the sensitivity of the dynamic behavior of CPU driven rations on the particular network implementation. It was our experience that the value of CPU target that leads to CPU driven flow control, as opposed to transceiver driven flow control, was sensitive to small changes in the protocols involved in congestion control and link-layer functions. However, these values were in the range of 0.7 to 0.8, and not in the range 0.9 to 1.0 where we feel the appropriate target values are likely to fall. We feel that faster processors should be an easy solution to processing bottlenecks, making the need for monitoring CPU resources redundant.

A simple approach to obtaining and idea of the traffic volume expected to produce CPU saturation is to assume equal traffic from each of the forwarding neighbors in a cross-traffic situation like the one simulated. Computing or measuring the amount of processing time per packet, as required by the node protocols, and computing the average packet interarrival time to the node as a function of aggregate traffic leads to a load factor equal to the ratio of this two quantities. For stable, feasible operation this load factor will have to be smaller than one. In practice the ability of the CPU to approach a load factor close to one will depend on its ability to multiplex service for cross-traffic streams.

For the network being simulated, the processing times associated with a packet were as follows (see Appendix A): preparing ack for a packet (13ms), computing packet route (4ms), processing ack received for a packet (1ms). The total time is 18 ms which translates into a maximum rate of 56 packets per second, uninterrupted CPU operation. The CPU use, given the carried load through the node in packets per second, is: *carried load*/ 56. The time average packets-per-second ration value, over the interval 20 to 150 seconds, for the experiments using *target*=0.75 gives 2.28 packets per second. We use this value as the steady state flow from any given neighbor into the cross-traffic node, for a total throughput of 16 · 2.28 = 36.48 packets per second through the node. This yields a CPU load value of 36.48/56 = 0.65. The discrepancy between this value and 0.75, the target value enforced by CPU rations, may be due in part to the statistical traffic fluctuations and oscillations observed.

4.4.4 Output Link Throughput

Transmissions on a link can be viewed as a first-come first-served service of throughput determined by the spacing and retransmissioni algorithms. Recall that we are interested only in the limitations on link throughput induced by channel noise. Since the parameter selection slgorithm strives to maintain a packet error probability on the link below 0.1, the spacing algorithm is the main mechanism determining the output link throughput, and not the retransmission algorithm. The two main components of the spacing wait are one moment of silence interval and the packet duration on the link. For small packets the spacing value is relatively insensitive to packet sizes and even to the choice of FEC or channel bit rate, since the resulting packet duration is comparable to the moment of silence interval. For relatively large packets the spacing wait, and hence the link throughput, become more sensitive to packet duration overhead on the link. In SURAP 4, the values of spacing wait can thus vary from around 50 milliseconds to about 100 milliseconds, reducing the throughput by a factor of two. A given flow through the link can go from feasible in the first situation to unfeasible in the second situation. If this flow is not individually

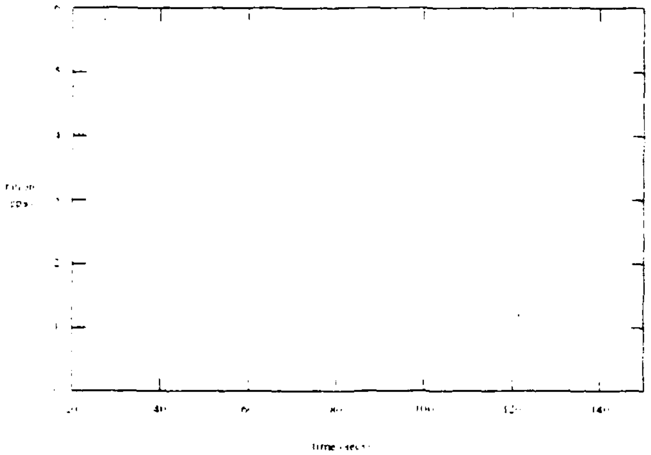


Figure 4.17: STEADY STATE PPS RATIONS FOR CPU TARGET = 0.9
The packet rations return to maximum value after the transient.

throttled back to a feasible value, buffer use measurements will eventually have to throttle all flows through the node – an unfair situation.

We are still gaining experience on the operation of output link rations. We have observed the problem described above in a simulation run in the OPNET™ simulator. We simulated three packet flows of 800-bit constant packet length crossing at a common relay node. All input and output links operated at 400 Kbps and without FEC[2], except for one output link which was set to operate at 100 Kbps and 1/2 FEC rate. The sources were sending through the node as fast as the network protocols allowed them, and the simulation was run without implementing output link rations. The rest of the congestion control algorithm was in place. The result was that a backlog build up in the K-Buffering queue for the flow using the 100 Kbps link, and this led to unnecessary throttling of all other sources. In fact, it was often the case that this flow was link-level flow controlled, and its throughput suffered considerably. At the time of writing this report we are still running more experiments in this simulator. We do not have enough data ready to report conclusively those results in this report.

A similar network situation was run in the Lisp-based simulator. This time, packet sizes obeyed the truncated exponential distribution already described. The topology used appears in Figure 4.18. Sources A and B attempt to send at a rate of 15 packets per second to nodes D and E, respectively, and their mean packet length is 500 bits. Source C attempts to send at a rate of 20 packets per second, of mean length 1500 bits, to node F. The output link from the merge node in the path to F uses 1/2 FEC and 100 Kbps channel rate. All other links operate with no FEC and 400 Kbps channel rate.

First the simulation was run without implementing output link throughput rations in the congestion control algorithm. Figures 4.21, 4.22 show the bits-per-second and packets-per-second ration values for all sources in the network. All sources are being throttled by the bits-per-second rations based on transceiver load. Oscillations appear at the beginning and just before the end of the simulation. We believe that the combination of large packets in the 100 Kbps output link and of the variable packet sizes are the cause for making the transceiver of the merge node the bottleneck. Since the ration computation interval is longer for the buffers, bits-per-second rations seem to be able to regulate the flow so that buffers do not become a bottleneck.

The experiment was run again, this time implementing the output link throughput rations. The rations are shown in Figures 4.19,4.20. This time the output link rations throttle the flow for all sources. The packets-per-second

[2] Even though the parameter selection algorithm calls for rate 7/8 FEC or lower at all times, we have chosen a "no fec" situation as more representative of the range of parameters the algorithm should be able to cope with.

rations for source C hover around 15 packets per second. The packets-per-second rations for sources A and B remain around 23 packets per second part of the time. Occasionally their ration values lower, together with that of C. During these periods the rations seem to be driven by buffer computations as can be seen by noting the ration update interval that applies during these periods. The throughput allowed for source C in this situation is almost the same as the throughput C obtained in the experiment run without output link rations. Simulation outputs showed a throughput of about 11 packets per second in both situations. However sources A and B are allowed unrestricted 15 packets-per-second flow in the simulation running with output link rations and the resulting total throughput is higher than in the simulation not using output link rations. We are still processing data on these and other simulations on output link throughput rations. At the moment we believe that the behavior in these two simulations presented showed the benefit of implementing output link throughput rations. The output link throughput rations seem to have prevented both buffer backlogs and high channel contention, freeing the rest of the resources for the use of flows from A and B.

Periodic instabilities are observable in the two figures discussed. This instabilities appear less frequent when output link throughout rations are at work. It is possible that the instabilities result from the fact that without output link throughput rations and a link working at such low throughput the service time in this link is subject to ocassional congestion leading to high buffer use and drastic throttling actions. We are currently investigating the reasons for these instabilities.

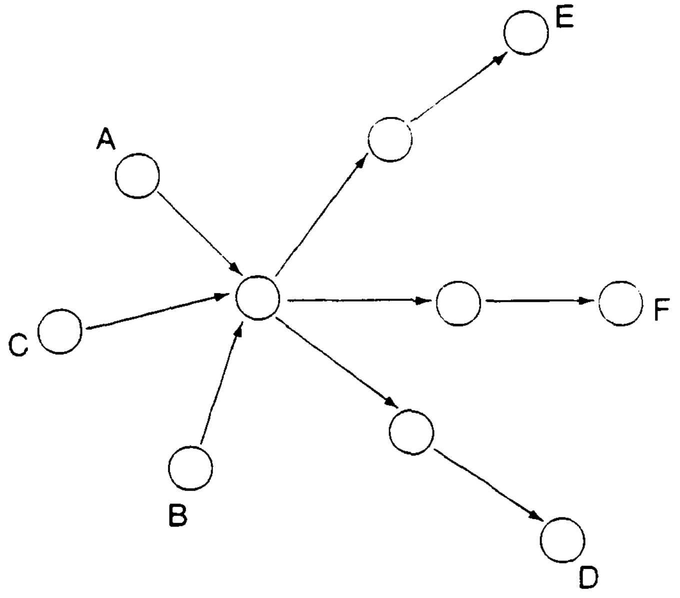


Figure 4.18: OUTPUT LINK RATIONS TOPOLOGY

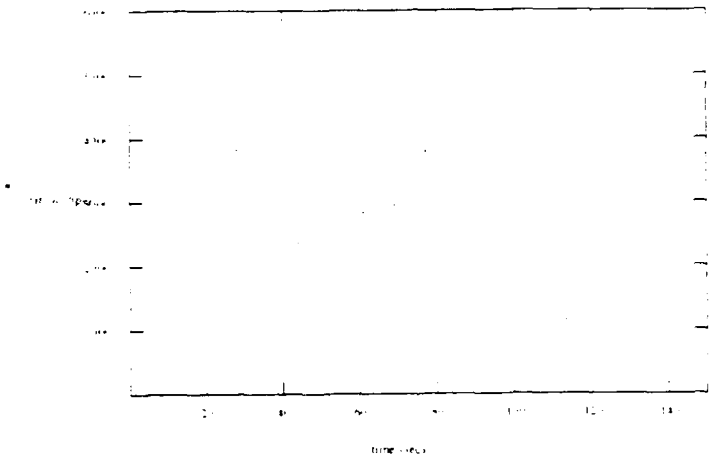


Figure 4.19: OUTPUT LINK RATIONS ON

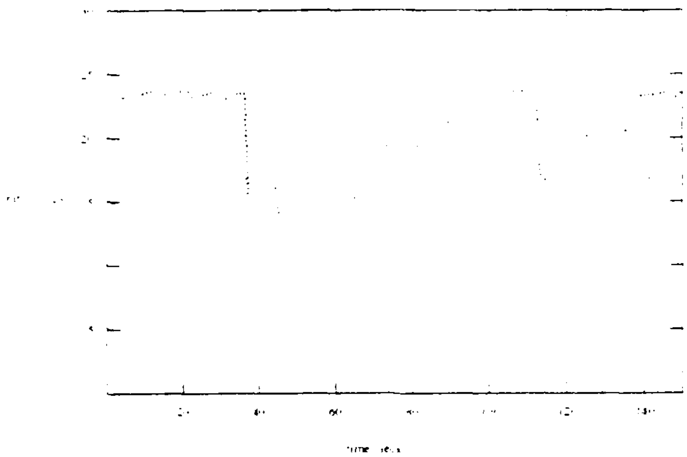


Figure 4.20: OUTPUT LINK RATIONS ON

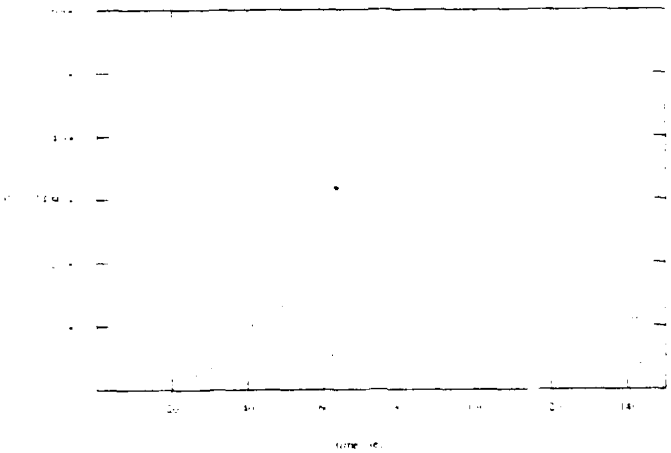


Figure 4.21: NO OUTPUT LINK RATIONS OFF

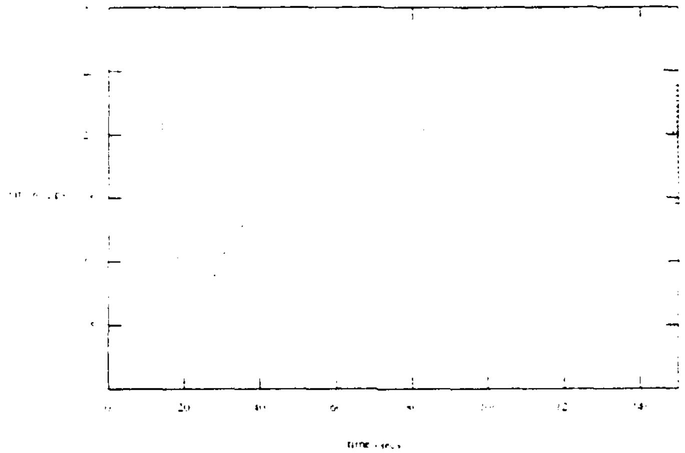


Figure 4.22: NO OUTPUT LINK RATIONS OFF

4.5 FAIRNESS

The set of end-to-end flows in the network is max-min fair if no flow can increase without forcing a decrease in another flow with already lower value. A centralized max-min fairness flow control algorithm starts with all end-to-end flows set to zero. It increments all flows equally until a resource bottleneck is reached: for example, some maximum node or link throughput. All flows sharing this bottleneck are held constant and the rest are increased uniformly until another bottleneck is reached. This is repeated until all bottlenecks are exhausted. A dual of this algorithm starts with excess flows instead of zero flows and decrease flows instead of increasing them.

Two factors complicate the implementation of a max-min fair flow control algorithm in a packet radio environment. First, it is hard to characterize capacity for nodes and links, as explained in Section 3.3. We have used instead resource load targets. Second, these capacities are not independent of the state of other nodes and links. There is coupling between node and link throughputs, due to the shared-media access nature of the network. Algorithm performance regarding the max-min fair operation, with respect to the main resources selected (transceivers and buffers), is the focus of this section.

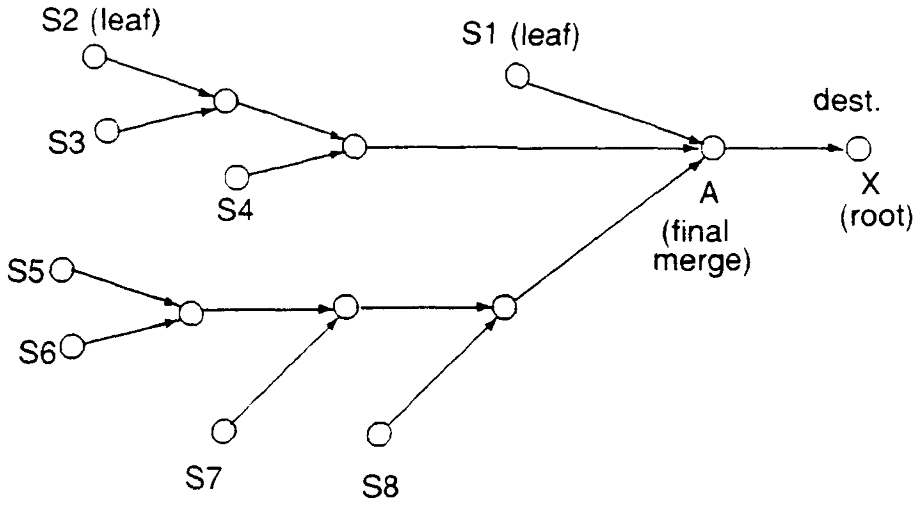


Figure 4.23: TREE TOPOLOGY Arrows stand for bidirectional links but point in direction of data traffic flow.

4.5.1 Transceiver Driven Rations

In exploring fairness we first explore the factors contributing to a transceiver bottleneck. Then, through three examples we consider how the appearance of bottlenecks relates to max-min fairness in the algorithm. We consider the flow situation in a tree topology, as in Figure 4.23, where traffic progressively merges to a *final destination* X. *In* this document we adopt the convention that, in the illustration of network topologies, arrows stand for bidirectional radio connectivity and point in the direction of data flow, as opposed to acknowledgement flow. Node A represents the final merge point. All sources are at the leaf nodes with no traffic generated at intermediate nodes. The effect of cross-traffic to other destinations, of the parameter selection algorithm, and of traffic generated at intermediate nodes is considered at the end of this section.

We assume that all sources have equal flow λ. The motivation is to explore which node becomes the bottleneck node and to check if the algorithm leads to max-min fair flows in the network. The bottleneck can be found by evaluating, among all nodes in the tree, the expression

$$\max\left\{\sum_i \lambda_i + \sum_i \lambda'_i r'_i\right\} \qquad (4.3)$$

except that the effect of the number of neighbors and traffic distributions, ignored for the time being, should also be taken into account.

For each node, we can define x to be the number of sources whose traffic flows through the node. Nodes transmit data only on one output link[3], namely the link en route to the destination. For a given node, $\sum_i \lambda_i = x\lambda$ and $\sum_i \lambda'_i r'_i = x\lambda r'$, where r' is the average transmissions per packet for the output link in use. Hence (4.3) reduces to

$$\max\{(1 + r')x\} \qquad (4.4)$$

This equation illustrates the relative impact of two important factors contributing to creating a transceiver bottleneck. The factor x in (4.4) indicates that nodes with the most traffic flowing through them are likely to

[3] From the model's point of view, we assume that the effect of alternate routing, when it occurs, can be represented by a change in the value of r'_i and hence does not alter the model.

become the bottlenecks. This is the basis for the heuristics of achieving max-min fairness using resource load metrics. The factor r' underlines the relevance of the channel access dynamics on fairness in this heuristic. It acts as a coupling factor between traffic processes in two neighboring nodes. Traffic flowing through a neighbor of a node may force this node to become a bottleneck because of the channel access interactions between them. However, expression (4.4) is more sensitive to x, a measure of traffic volume through the node, than it is to r', which represents channel coupling.

Let node A be common to the path from all sources to the destination. Let node B be any upstream node from node A. Since traffic flow through A must be at least as large as through B, $x_A \geq x_B$. Alphabetical indices will be used to indicate the node being considered. Equality holds in a situation with no further traffic merges between nodes B and A, for example, if the tree is simply a line of nodes. This case is uninteresting since we can do away with all nodes between B and A, including B itself, as far as exploring bottlenecks is concerned, without loss of generality. In the interesting case $x_A > x_B$, for any node B to be considered a bottleneck over node A it would be necessary, from (4.4), that

$$r'_B > \frac{x_A}{x_B}(1 + r'_A) - 1 \quad . \tag{4.5}$$

This expression does not presuppose an accurate model for average number of retransmissions to a node. The identity of the bottleneck in a given flow pattern is shown to play a role in achieving fair flows in the network. Mean packet length of 2500 bits was used in the simulations treating transceiver driven rations. The packet length distribution was the truncated exponential distribution described in Section 4.2. The values for minimum and maximum packet length chosen was 224 bits and 3000 bits, respectively. The impact of the number of neighbors and traffic distribution over the input links is unlikely to be relevant in the first two of the three following examples.

Case 1 (homogeneity): The intention of this example is to explore the effect of traffic volume on fairness. We choose a homogeneous network as illustrative example. Consider a tree topology such that $\phi_I = \phi \geq 2$ for all intermediate nodes I, that is, all nodes *within* the tree have the same number of branches, except for the root and leaf nodes, as shown in Figure 4.24. Leaf nodes are the only traffic sources and the root node X is their common destination. Node A is the *final merge* node for the data traffic flow, and is the only upstream neighbor of node X. Thus $r'_A = 1$. In this sense node A differs from the rest of the intermediate nodes. Inequality (4.5) gives

$$r'_I < 2\frac{x_A}{x_I} - 1 \qquad . \tag{4.6}$$

From the symmetry among nodes in the network, $x_A \geq \phi x_I \geq 2x_I$. Hence, with the target limit of $1 < r \leq 1.5$ chosen for the algorithm, and traffic evenly distributed among input links to nodes, it is unlikely that any intermediate node B will satisfy (4.6). Node A, the final merge, becomes the bottleneck and all sources receive equal rations through backpropagation, maintaining the initial condition of equal traffic rate from all sources, which corresponds to max-min fair steady state flow on the network.

The tree need not be completely symmetric. Figure 4.25 plots the bit-per-second rations for all source nodes in the network of Figure 4.23, showing convergence to equal steady state flows. This case corresponds to max-min fair flows. Packet-per-second rations do not participate in flow control.

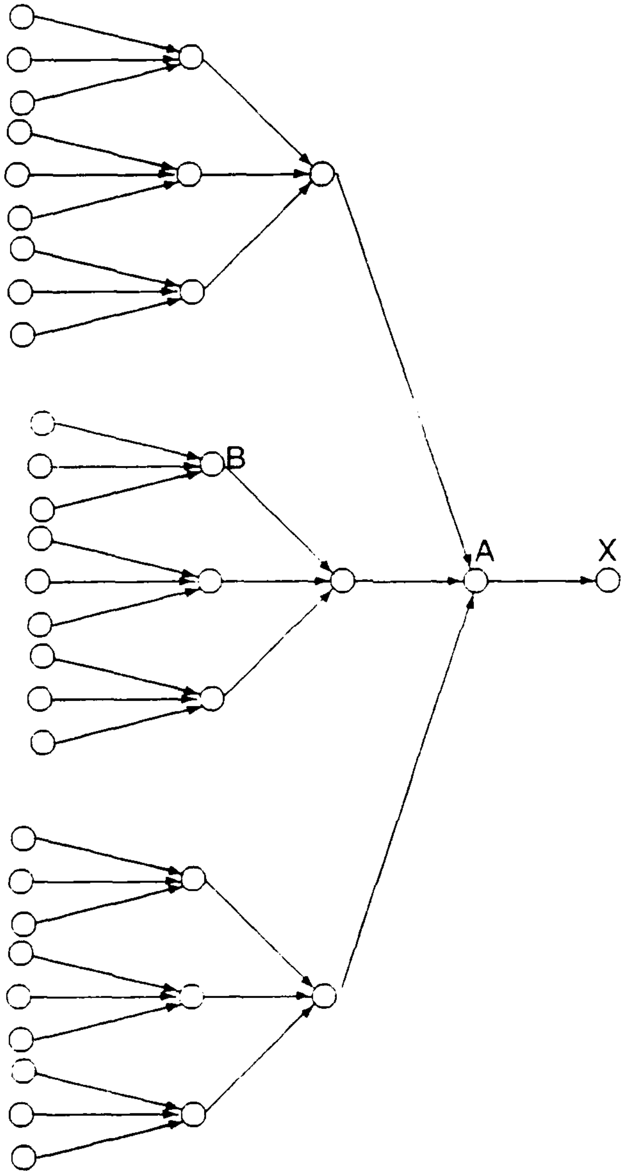


Figure 4.24: HOMOGENEOUS TREE TOPOLOGY

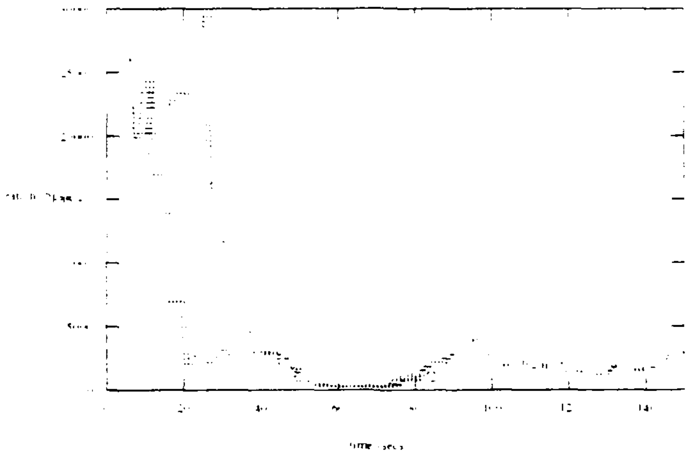


Figure 4.25: RATIONS FOR TREE TOPOLOGY (LARGE PACKETS)Fair flow achieved.

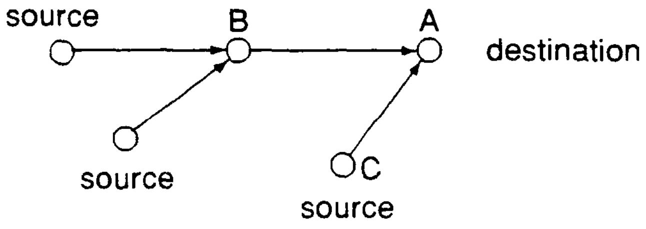


Figure 4.26: TRAFFIC MERGING AT DESTINATION

Case 2 (retransmissions): The intention of this example is to illustrate the effect of channel access on fairness. To dramatize the effect of retransmissions, we consider a case where the final destination is node A, such that $r'_A = 0$, and let $x_A = x_B + 1$. An instance of this situation is illustrated in Figure 4.26. Node A is the destination node and nodes B and C are one hop upstream from node A. Because the destination node does not relay packets then the average number of transmissions per packet into this node is typically lower, for a given throughput, than for node B. In particular, inequality (4.5) becomes

$$r'_B > \frac{1}{x_B} \quad . \tag{4.7}$$

Since $r'_B \geq 1$ and $x_B \geq 1$, B is certain to become the bottleneck, if the transceiver model is representative. Path rations for node C are determined by node A while path rations for the other two source nodes are determined by node B. The path rations for C are greater than or equal to the path rations for the other two flows. It is fair that flow from C have higher path rations than the remaining end-to-end flows since node B experiences more contention than node A. However, r'_B, the average transmission per packet from B to A, depends on the carried load from C to A. A problem regarding fairness now arises. Once node B becomes the bottleneck it throttles its traffic and node A allows a grater flow from node C. This very flow increases r'_B and further penalizes the flow through node B, which runs counter to the definition of max-min fairness. This situation was identified in simulations and constitutes a valid class of exceptions to the max-min fair operation of the algorithm. The example appears in Figure 4.27. The ration that remains at a constant 60,000 bit-per-second value applies to flow from node C. In contrast, without flow from C the flow from the remaining two sources can double. This shows that the flow from C is a major cause in limiting the flow from the other two sources, as explained. When we simulate a network where node A is a relay node to the final destination, as in Figure 4.28, the behavior of the rations is as shown in Figure 4.29. Inequality (4.5) yields $r'_l > 1 + 2x_l$, making it unlikely that B become a bottleneck unless B has four or more neighbors. The resulting fair flow allocation can be observed in the figure. The flow from C is limited considerably and becomes equal to the flow from the other two sources. The reason why the flow from these two sources is not larger on average than their flow in Figure 4.27 is as follows. The flow on the link from A to the destination in the network of Figure 4.28 consumes part of the transceiver throughput of node A. Effectively the network of Figure 4.28 has a smaller throughput to its destination than the network of Figure 4.26.

We have also simulated the behavior of the SURAP 1 architecture, which uses pacing as the main mechanism for flow control, using an Opnet™ based simulator. The Opnet™ simulator is being used by the Suran Project at BBN to benchmark the performance of SURAP algorithms. Full results will be documented in [6]. The simulation of the pacing algorithm on the network of Figure 4.28 shows that the steady state packet-per-second flow from node C is approximately twice as large as the flow from each of the other two sources; while the total network throughput is not greater than the throughput maintained by the SURAP 4 Congestion Control Algorithm.

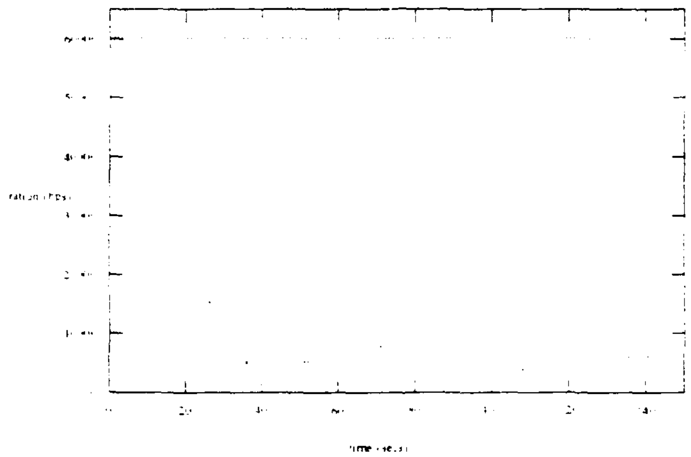


Figure 4.27: RATIONS FOR MERGE AT DESTINATION (LARGE PACKETS)
Unfair flow. Bit rations implement flow control. Path rations for source closest to destination remains at maximum value.

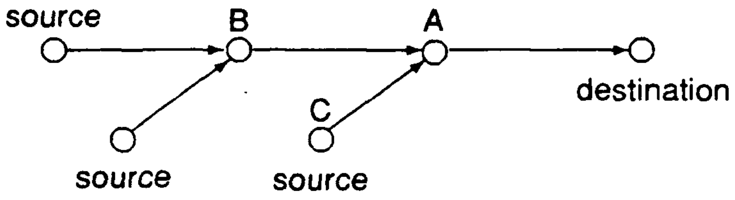


Figure 4.28: TRAFFIC MERGING AT RELAY

The fairness problem identified can be generalized as follows. Once a bottleneck appears, it reduces flow through itself, freeing resources at its downstream *neighbors*. If, consequently, the feedback to sources provided by the congestion control algorithm increases other flows to these downstream neighbors this action may increase the transmission component of contention at the bottleneck via higher offered load (retransmissions) in its output links. The bottleneck further throttles its traffic, reinforcing the trend (positive feedback) and worsening the situation. This can constitute an unfair flow situation. The problem is not the identity of the bottleneck (node B in the example), but rather the excessive throttling induced by the coupling factor r' between the bottleneck and its downstream node. This "excessive throttling" behavior is a risk whenever the cause for a bottleneck is something other than traffic volume. This includes cross-traffic situations and situations with many neighbors sending to a node.

For a cross-traffic situation, consider a node with traffic flowing to a destination X, using one output link, and cross traffic flowing to another destination Y, using a different output link. Traffic to Y can turn the node into a bottleneck for traffic to X, in a situation like the one described in this second example. If traffic to X and traffic to Y are *comparable*, traffic to X through the node is a contributor to retransmissions at the neighbor downstream on the path to X. An unfair situation like the one described follows. If traffic to X is significantly *smaller* than traffic to Y, changes in flow to X are unlikely to have a noticeable effect on the ration computations at the bottleneck node. If traffic to X is significantly *larger* than traffic to Y, it is unlikely that the cross traffic will turn the node into a bottleneck for traffic to X in the first place.

The next example illustrates a situation where the bottleneck is brought about by a large number of neighbors sending to a node.

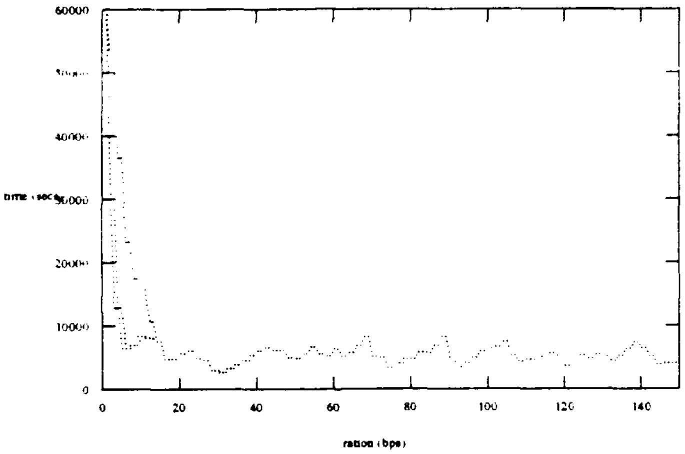


Figure 4.29: RATIONS FOR MERGE AT RELAY (LARGE PACKETS)
Fair flow. Bit rations implement flow control.

Case 3 (neighbors): The intention of this example is to explore the effect of the number of node neighbors on fairness. We consider a situation with node B an upstream neighbor of the final merge node A, and ϕ_B large but $|\phi_A|$ small. That is, node B receives flows from many neighbors while node A receives flows from only a few neighbors. An instance is illustrated in Figure 4.30. Heuristically we modify the arguments in expression (4.4) to $(1 + r'_B)x_B$ for node B and $(a_A + r'_A)x_A$ for node A, where $a_A < 1$ reflects the fact that, for a large enough ϕ_B , traffic forwarded to node A is expected to experience lower contention than traffic to node B. From the figure, without loss of generality, $x_A / x_B \simeq 1$ and $r'_A = 1$. Inequality (4.5) reduces to

$$r'_B > a_A \; ; \tag{4.8}$$

a certainty. The fact that traffic into node A is skewed while traffic into node B is uniformly distributed among links is not expected to overcome the problem for large enough $|\phi_B|$. The results of ration values for the network of Figure 4.30 are presented in Figure 4.31 and seem to confirm this conjecture. The consistently higher path rations corresponds to source C. The consequence of node B becoming the bottleneck in this situation is similar to the problem discussed under case 2. The effect of traffic from C can be seen in Figure 4.32 which repeats the experiment omitting traffic from C. For the first instance (flow from C, Figure 4.31), an average of the path ration values for flows through node B, taken from time 50 secs to time 150 secs (steady state) yields values of 855 bits per second, plus or minus 50 bits per second. In the second instance (no flow from C, Figure 4.32), the same average computation yields 1520 bits per second, plus or minus 20 bits per second for the flows through B.

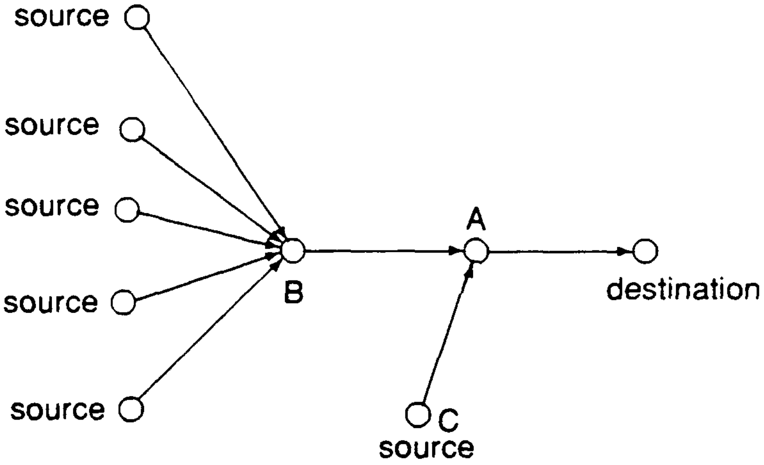


Figure 4.30: TRAFFIC FROM MANY NEIGHBORS

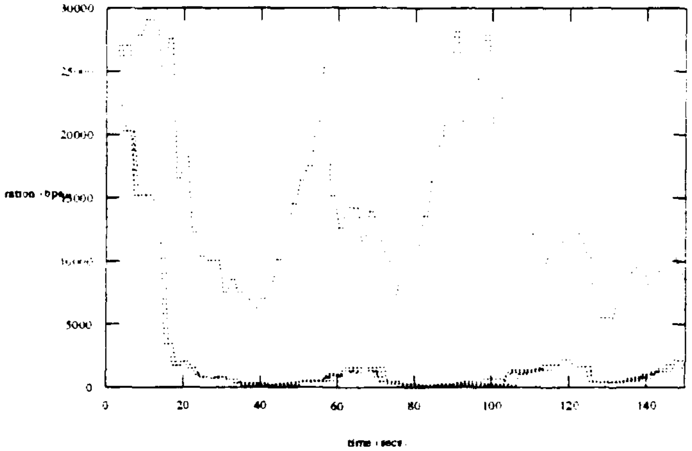


Figure 4.31: RATIONS FOR MANY NEIGHBORS CASE (LARGE PACKETS)
Unfair flow. Bit rations implement flow control. Path rations for source closest to destination are consistently the highest.

These three examples are presented as paradigms of fairness operation in the algorithm. They have been simplified to facilitate the explanations. We now reconsider the effect of variable packet length, the parameter selection algorithm, cross traffic, and host traffic at intermediate nodes. The effect of all of them is merely to affect one or another component of contention in (4.3) and hence determine which node becomes the bottleneck. The comments made under the three examples still hold when applied to such a bottleneck.

Variable packet length traffic is likely to affect computation of $< r >$ primarily through its average value. In this sense the average packet duration could be replaced for t in expression (4.3) and the argument that $< r >$ is monotonically increasing in this expression would remain valid. The steady state operation of the parameter selection algorithm may lead to different expansion factors for the packet duration in each directed link. These can be included as factors into each of the components in (4.3), link by link. Cross-traffic and host traffic at a node add extra carried load through the node. Traffic from the host only affects the transmission component in (4.3) while traffic destined to the host only affects the reception component.

4.5.2 Buffer Driven Rations

The choice of *target* for buffer use is a compromise between acceptable throughput when receiving from many neighbors, that is, allowing high $\overline{n}$ if x is large, and preventing the node from becoming a storage facility when receiving from only a few neighbors, for example, $x = 1$. The choice of $K = 9$ for K-Buffering, on the other hand, reflects two goals. First, to make K large enough so that the primary throttling mechanism is the ration-based congestion control. Secondly, that K is not so large that a single neighbor forwarding can monopolize all buffers.

For the discussion in this section again we focus on a merging traffic situation with all traffic flowing to the same final destination, as illustrated previously in the tree topology of Figure 4.23. This implies that nodes have only one output link for packet transmission. Once again we analyze a situation where all sources have equal traffic λ. From expression (4.2) (Section 4.4.2), which gives the number of buffers in use as a function of the load factor in an M/G/1 queue, the bottleneck in the tree can be identified as that node for which the load factor ρ is highest:

$$\max\{\lambda' s(r')\} \ . \qquad (4.9)$$

Maximization is over all nodes in the tree. If the maximum corresponds to a node common to all flows, every source receives the same ration value and steady state max-min fair flow is maintained. As in Section 4.5.1, we use x to denote the number of sources whose traffic flows through the node. Thus (4.9) can be expressed as

$$\max\{x \ s(r')\} \ . \qquad (4.10)$$

The situation is similar to that expressed by equation (4.4) just that this time r' has a larger effect on determining the bottleneck. We look into the same three cases as for transceiver utilization. Mean packet length of 500 bits was used in the simulations treating buffer driven rations. The packet length distribution was the truncated exponential distribution described in Section 4.2. The values for minimum and maximum packet length chosen were 224 bits and 3000 bits, respectively.

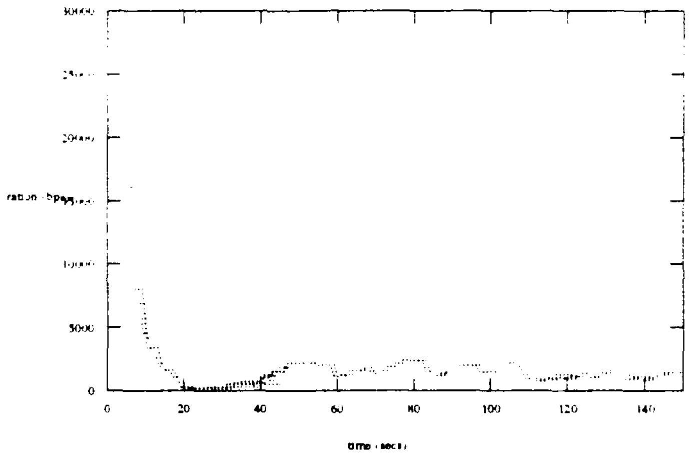


Figure 4.32: RATIONS FOR NEIGHBORS CASE WITHOUT OUTPUT CONTENTION (LARGE PACKETS)
Flow decrease from source C allows higher path rations for remaining flows.

Case 1 (homogeneity): In the homogeneous tree network of Figure 4.24 max-min fairness is expected. The number of retransmissions is expected to be low for all output links when small-length packet traffic predominates, and so transceiver contention is likely to be low. Thus the main factor contributing to a bottleneck is the traffic volume through a node. Figure 4.33 plots the packet-per-second rations for all source nodes in the tree topology network of Figure 4.23 when average packet length for the traffic is small.

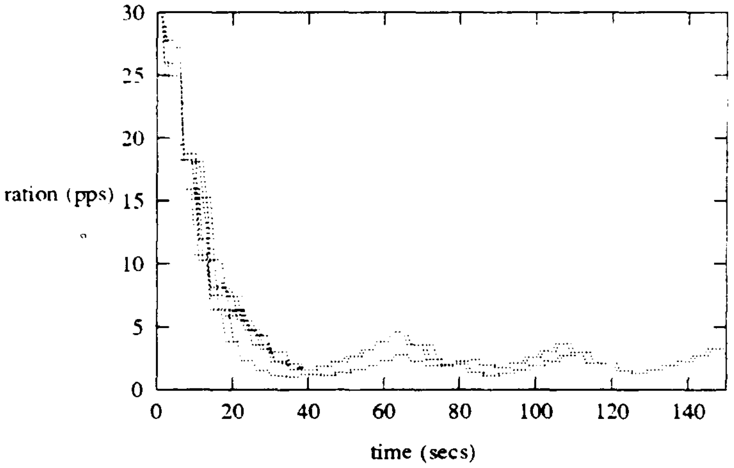
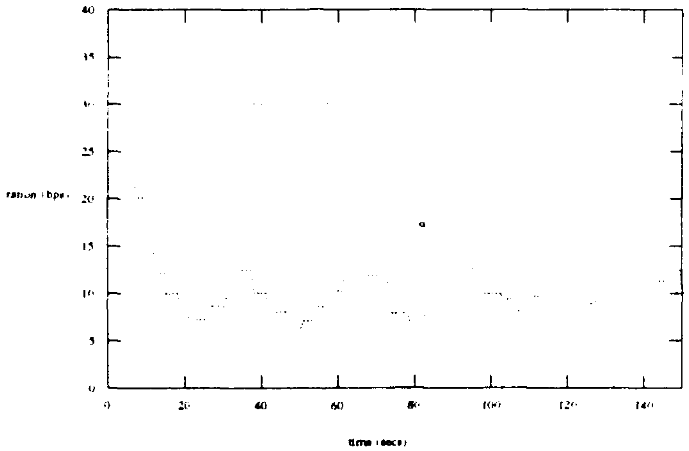


Figure 4.33: RATIONS FOR TREE TOPOLOGY (SMALL PACKETS)
Fair flow. Packet rations implement flow control.

Case 2 (retransmissions): If a situation similar to that of Figure 4.26 is considered, where the final merge node A is also the traffic destination, then node B, the upstream neighbor of A, is expected to become the bottleneck. The same problem described under case 2 in the previous section (positive feedback to traffic from node C via the congestion control algorithm) is expected to arise, excessively penalizing traffic through node B. This has been observed in simulation results, as shown in Figure 4.34. Again, when node A is a relay node the problem is unlikely to arise. Figure 4.35 shows the ration values for this last situation.



Fig^'e 4.34: RATIONS FOR MERGE AT DESTINATION (SMALL PACKETS)
Unıaır flow. Packet rations implement flow control. Path rations for source closest to destination remain at maximum value.

In general however, even when the final merge node, A, is itself a relay node, if

$$s_B(r'_B) > (x_A/x_B)\, s_A(r'_A)\,, \tag{4.11}$$

where index B refers to some upstream node, node A is unlikely to become the bottleneck. Throttling by the bottleneck node B could lead to increased traffic through A for flows not coming through B. If node B is also an upstream *neighbor* of A, this could lead to increased retransmissions in the output link of node B and worsen the situation further, leading to unfair throttling at B.

Case 3 (neighbors): The case of many neighbors forwarding to a node is not expected to contribute as significantly to making this node a bottleneck as in the transceiver utilization case. Figure 4.36 plots packet-per-second rations for the experiment with several neighbors of Figure 4.30 but run with small packet lengths. In this case it is the packet-per-second rations which drive the flow control process instead of the bit-per-second values as in case 3 for transceiver bottlenecks. The ration values for source C however are still consistently higher than the rest, as the flow rations plotted show. The reason for this is found in equation (4.11) since the ratio x_A/x_B is 6/5 = 1.2 and node A experiences no output retransmissions. This is thus an example of case 2 above and not a consequence of the number of neighbors forwarding to node B.

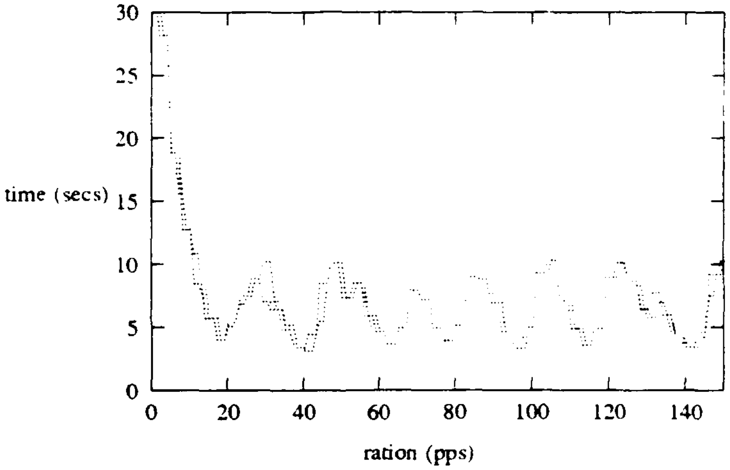


Figure 4.35: RATIONS FOR MERGE AT RELAY (SMALL PACKETS)
Fair flow. Packet rations implement flow control.

4.5.3 Alternative Schemes

The value of K suggested above is close to the value N, the number of buffers available per node. In Section 2, we explained that the motivation for K-Buffering is to limit the buffers in use by a single neighbor. From this point of view, and considering that nodes are not meant to be storage facilities, it seems reasonable to implement a lower value of K. In a scheme using a target buffer target value BUFFER_TARGET to regulate packet end-to-end packet flow there exists, however, a compromise among three elements: the value of K, fairness, and network throughput. The relationship between $\overline{n}$, the number of buffers in use at a node, and throughput through the output link of a node is given by Little's formula $\lambda s(r') = \overline{n}$. This means that for a fixed average service time in the link, the number of buffers in use is proportional to the carried load through the link. The relationship between $\overline{n}$ and fairness can be found through the backlogs $\overline{n}_j$ corresponding to neighbors sending to the node. In particular, if $\overline{n}_j$ becomes close to K while $\overline{n} <$ BUFFER_TARGET then it is very likely that K-Buffering will throttle the flow from neighbor j with no attempt at preserving max-min fair flow. Under the assumption of infinite buffers, $\overline{n}$ and $\overline{n}_j$ are related by $\overline{n}_j = \left(\lambda_j / \sum_{\phi} \lambda_j\right) \overline{n}$. Even though the infinite buffer assumption is not realistic, we expect that if

$$\lambda_j / \sum_{\phi} \lambda_j > K/\overline{n} \qquad (4.12)$$

then the average number of buffers stored for node j will be dangerously close to the K-Buffer limit. Given a small enough K and a large enough BUFFER_TARGET, this situation could happen if many end-to-end sessions are multiplexed in the traffic from j while only a few in the rest of the traffic to the node. In this situation, with all end-to-end traffic increasing from zero, the K-Buffering limit for neighbor j becomes a constraint for that flow before any end-to-end congestion control detects a problem. The rations then continue to increase and so do the end-to-end flows not using neighbor j. This is unfair since sessions not using neighbor j are being allowed to increase at the expense of the sessions through j. Notice that the problem can be avoided by choosing BUFFER_TARGET to satisfy BUFFER_TARGET $< K$, c.f. equation (4.12). In this case, even if all traffic is coming from a single neighbor j the target value for end-to-end congestion control is lower than the K-Buffering limit. Now, with BUFFER_TARGET $< K$ for fairness considerations, and Little's formula $\lambda s(r') = \overline{n}$, it follows that low K implies low throughput, especially in a situation with many neighbors sending to the node and end-to-end sessions through the node evenly distributed among the neighbors forwarding to the node. For this reason we recommend the operation with relatively high values of K and a value BUFFER_TARGET small enough so that the variance of the flow process seldom triggers K-Buffering throttling and so that the nodes do not become storage facilities with the inevitable high delay. As mentioned before, we loose some of the shielding effect desired from K-Buffering. For the LPR's in the SURAP network the total number of buffers is $N = 12$ and we recommend $K = 9$, BUFFER_TARGET $= K/2$.

Because the queue size, the queuing delay, the queue-full probability, are all monotonically related in steady state, the same argument made above using the queue size $\overline{n}$ could have been made using the other two quantities, if these were chosen as the mechanism for flow control.

We now discuss one approach aimed at keeping low values of K and still maintaining fairness characteristics which seems promising at first sight, but on closer scrutiny exhibits problems. The intention is to record some of the knowledge accumulated during our work in congestion control. Consider the following algorithm for setting the ration values based on buffer utilizations.

1. Compute the time average buffer use for each neighbor sending to the node, over the ration computation interval.

2. Compare the *maximum* buffer use, computed among all neighbors, to a target value aK, with $0 < a < 1$. If the use is smaller increase the ration; if it is larger decrease the ration.

Because this scheme works with the queue with maximum use (maximum-backlog queue), the end-to-end congestion control always acts, in steady state, before K-Buffering limitations come into permanent play. When all sessions

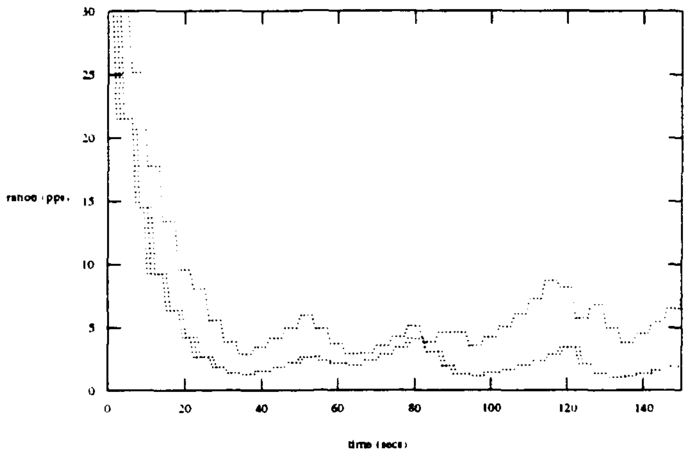


Figure 4.36: RATIONS FOR NEIGHBORS CASE (SMALL PACKETS)
Unfair flow. Packet rations implement flow control. Service time at A is shorter than service time at bottleneck B. Flow from source closest to destination is consistently the highest.

through the bottleneck node have a flow equal to the ration value, the K-buffering backlog corresponding to the input link serving the greatest number of sessions is close to aK, while backlogs with a smaller number of sessions end up being smaller. Thus the total buffer use adapts to the session traffic pattern through the node.

Figure 4.37 shows the packet-per-second rations observed in a simulation running this method of buffer control on the same topology used for the example where the final merge is a relay node (Figure 4.28) with traffic of small mean packet length (500 bits). The experiment used $K = 3, a = 0.6$. One ration consistently dominates the rest and corresponds to that of the source through node C. The problem with the scheme appears to be that the variance of the buffer process is too large for the small difference between the *target* value aK and the maximum value K and for the value of measurement intervals that seem appropriate for fast convergence of rations. For reasonable throughput and small values of K the difference between K and aK is likely to be less than two buffers. If the K value is reached, the forwarding delay wait imposed by K-Buffering suppresses traffic to the queue, for most of the measurement interval by backloging the queue in the upstream neighbor. The problem seems to result in a random interchange of bottleneck, from node A to node B, biased to make B the bottleneck most of the time, perpetuating the unfairness of the situation. Indeed, choosing a longer measurement interval in simulation experiments seems to provide longer spells of "fair" operation, but the problem still persists.

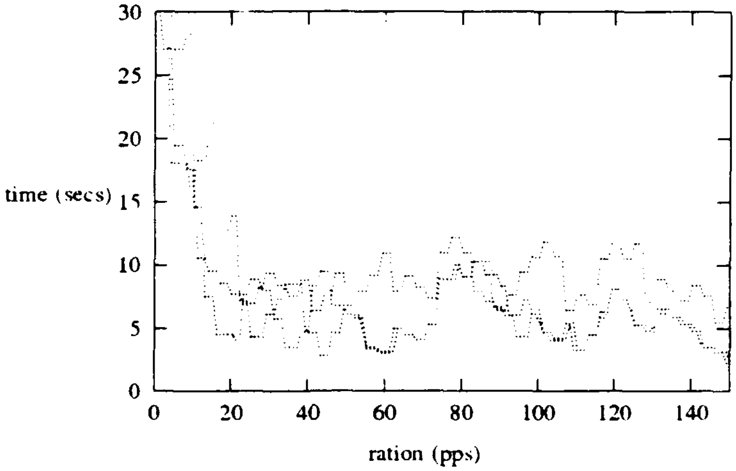


Figure 4.37: RATIONS FOR MAXIMUM-BACKLOG QUEUE METHOD (SMALL PACKETS)
Unfair Flow. Packet rations implement flow control. Path rations for source closest to destination remain consistently higher.

The problems in achieving max-min fairness discussed throughout Section 4.5 could be avoided if a reasonably accurate model of resource use were available. In particular, a model of the dependence of the number of transmissions per packet forwarded to a node as a function of the load through this node and a model of the dependence of output link service time on the load through the link and the load through the receiving node would allow to factor out the effect of retransmissions for the output links of a node from resource use computations. Such models are hard to treat and have not been incorporated in our algorithm.

4.6 CONVERGENCE

In this section we address the question of the time lag between a bottleneck developing and the moment the algorithm imposes a steady state flow rate sustainable by the network.

Number of Iterations As a characterization of the number of iterations required by the algorithm, we focus first on the pathological yet important situation when a bottleneck develops due to a sudden onrush of traffic. We discuss the behavior of the algorithm through some of the simulations discussed earlier in the document. We begin by looking at the convergence of bit-per-second rations in the tree topology of Figure 4.25. In the simulation, initially all sources sent traffic at a rate of 20 packets per second; the mean packet size was 2500 bits for all sources. The bit-per-second rations implement the flow control as discussed in Section 4.4.1. The number of sources sharing the bottleneck is eight, so we expect convergence to a flow about one eighth the maximum throughput of a node. Figure 4.25 shows that one eighth of the maximum throughput achieved corresponds to about 1600 bits per second, implying a throughput of about 5 packets per second. With a priori uncertainty regarding the feasible value of source flows, one can argue that a binary search would be a reasonable strategy to converge to the correct value. We consider such a strategy as a means of assessing the convergence rate of our algorithm. Starting from a maximum ration value of 60,000 bit per second this strategy would, in an ideal situation (no overshooting, no instabilities), converge to within 10%, say, of this value in 8 iterations, leading to a convergence time of 8 ration measurement intervals (2 seconds), or about 16 seconds. From the figure we observe a convergence lag time for our algorithm of approximately 40 seconds. Applying the same reasoning to the convergence shown in Figure 4.29, which shows path rations for the network consisting of three sources and the final merge being a relay node (Figure 4.28), the rations converge to approximately 5000 bits per second, requiring about 6 iterations of a binary search starting at 60000 bits per second, for a total of approximately 12 seconds. In the figure the convergence lag time observed is approximately 16 seconds. Even though the figures illustrate the convergence times that can be expected with the algorithm, we are more interested in the dependence of this time on the size of the network.

The basis for the analogy between the binary search and the ration decrease dynamics is the fact that when node rations are near maximum value and traffic comes through at essentially the maximum, uncontrolled throughput permitted by the network, there is little correlation between measured load at the bottlenecks and the rations they compute, since traffic rates are not being limited by rations but by link protocols. Only when rations reach a value close to the final feasible source rates are the changes in ration values be reflected in the measured load at the bottlenecks. Hence, during the initial part of the process, rations, for the most part, decrease by the same factor over and over again. This observation applies equally to any of the three ration control laws stated in equations (3.1), (3.2), and (3.3) in Section 3.2.

The assumption of node rations starting at maximum value is relevant for a network characterization, since in the absence of traffic ration values drift from their current values to the maximum value due to low load in the nodes. The characterization serves then as a worst case scenario, but likely to occur in a tactical network environment. This leads to the conjecture that for large networks the number of iterations required is $\mathcal{O}(\ln S)$, where S is the number of sources expected to share a bottleneck in the network. In the examples illustrated, $\ln 8 / \ln 3 = 1.9$. The ratio of convergence times for the binary search is $16/12 = 1.33$ and the same ratio computed for the Congestion Control algorithm is $40/16 = 2.5$. A much greater number of simulations will have to be run to test this conjecture.

Notice that for a given network design, the convergence time is dependent on the size of the network mainly through the number of sources sharing the bottlenecks, since the ration measurement interval is a design parameter chosen large enough to accomodate the maximum size network expected.

Stability One other important aspect of the convergence behavior of the algorithm is its ability to adjust to gradual changes in load and its ability to mantain a stable steady state operating point. The ability to track changes in load depends on how large these changes are. If resources do not become completely saturated by the change in network load we expect a fairly rapid convergence, of the order of a few iterations. For resources where the load is linear in the flow, like CPU, convergence in about one or two iterations would be expected. Unfortunately use of the main resources, transceiver and buffers, is markedly non-linear in flow. From the simulation results discussed in previous sections it is apparent that the algorithm suffers from oscillatory behavior in steady state. The non-linear relationship between flows and resource load seem to be at least partly responsible for this behavior.

In Section 3.2 we stated three ration update control laws. These were the basic control law, the asymmetric control law, and the cube root control law of equations (3.1), (3.2), and (3.3), respectively. This equations were, in that order,

$$ration(t+1) = \min\{threshold, \frac{target}{load} ration(t)\},$$

$$ration(t+1) = \min\{threshold, \frac{target}{load} ration(t), 1.2ration(t)\},$$

$$ration(t+1) = \min\{threshold, \sqrt[3]{\frac{target}{load}} ration(t)\},$$

We noticed large oscillations in steady state when using the basic ration control law. This persisted even if large measurement intervals were used. The problem appeared to be periodic overshooting by the rations, which seemed unable to smoothly increase up to the appropriate value. The simulations presented in this report were instead run with the asymmetric control law, which limits the rate of ration increase. The amplitude of the oscillations were reduced to a point which seems to afford reasonable operation. After focusing on and solving other problems which we considered of greater importance, notably fairness, we are currently running more simulations focusing on the problem of oscillations. These simulations are being run on an Opnet^{TM} simulator package for broadcast networks. The results will appear in a future report. The conclusions to date are as follows.

Buffer use seems to pose the greatest problem regarding oscillations. In the case of buffer driven rations, the problem seems to be a combination of two main factors: first, the rate at which rations are increased, and second, the way in which buffer use is measured. The best results were obtained by using the cube root control law for buffer rations, and a time average buffer use measure instead of the average number of buffers in use upon arrival of a packet at the node. Neither of the two factors by itself was able to eliminate the oscillations in the test networks simulated. The two modifications used together allowed fast and smooth convergence to a stable, fairly constant value in a test network with greedy sources. Since all tests planned have not been completed at this moment we are reluctant to claim that the problem of oscillations has been solved, and fine tuning is still in progress. However, some of the conclusions appear solid.

The problem presented by the resource measurement based on the average number of buffers in use upon arrival of a packet seems to be as follows. As the flow through the node decreases, packet retransmissions and forwarding delays reduce to a point where the spacing algorithm is able to regulate in an almost deterministic fashion the transit of packets through the node, avoiding queueing delays. The situation is analogous to a queue with deterministic arrivals and low-variance service times. The number of buffers in use upon a new arrival is typically zero, regardless of arrival rate, until the use factor reaches a value close to one. At this point the queue size experiences a marked increase as the arrival rate increases. This abrupt change in characteristic seems to contribute to make the ration control law overestimate the ration increase required to achieve the target resource load. When the time average number of buffers in use over the measurement interval is used, the measured load is more or less linear in the arrival rate in the same situation, until the use factor becomes close enough to one and buffer use starts to take off. In simulations we have implemented this measurements by taking periodic samples and averaging over the number of samples in a buffer measurement interval.

Measurement Interval Apart from the iteration delay, the control delay in the congestion control algorithm has three more components:

1. Detection delay: between a bottleneck appearing and the next ration update in the associated node.
2. Backpropagation delay: between this update and the ration reaching all sources sharing the bottleneck.
3. Response delay: between sources enforcing the new flow ration and resources at the bottleneck reaching a new steady state loading.

The last two can be thought of as the feedback delay since they represent the delay in the network from the moment a ration is updated at a bottleneck node (the control input) to the moment the effect of this action is felt as a steady state change of resource load at the node (the state observation). The duration of the measurement interval in a network must be tuned to these three delays. Since the first delay grows linearly with the measurement interval, this should be as small as possible as permitted by the feedback delay.

The speed at which backpropagation communicates rations from bottlenecks to sources can be estimated as follows. Over path segments where traffic is flowing at a rate leading to buffer occupancies of one or more along its path, rations backpropagate with a delay of the order of an average packet forwarding time per node. This is because once the ration value arrives at a node it must wait for the next acknowledgement upstream, en route to the source. Since there is typically a packet waiting for transmission as soon as the previous one is acknowledged, in the worst case the delay is a packet forwarding delay from the neighbor upstream. Over path segments where traffic flow leads to average occupancies of less than one buffer along the path the backpropagation wait at a node includes. in the worst case, the time for a new packet to arrive at the neighbor upstream (inter-packet delay) plus the packet forwarding time from that same neighbor. Thus in this case, the wait is of the order of one inter-packet delay plus one packet forwarding delay, with the former more relevant for slower traffic. Interpacket delay however is not expected to be typically relevant since bottlenecks are likely to arise under heavy traffic[4]. The delay to reach the source is then the sum of the per node delays of the newly computed ration. Once this information arrives at the source and is enforced, there is a delay equal to the packet transit delay between source and bottleneck before the bottleneck feels the change has taken effect and recomputes the ration.

For large networks, the feedback delay is expected to be dominated by packet forwarding times at the nodes, which can be considered independent of network size, and the appropriate value of measurement interval is expected to grow in the product ND where N is the maximum size of network envisioned and D is of the order of an average packet forwarding delay in the nodes.

4.7 ROBUSTNESS

For proper operation, the congestion control algorithm needs nodes that are able to compute their rations faithfully, so that backpropagation to sources can take place, and sources can enforce path rations.

We first consider a situation with simple node and link failures such that nodes or links either function correctly or do not function at all. The ration computation algorithm in a correctly operating node does not depend on the functional state of neighboring nodes or adjacent links. The computation is done periodically based on measurements of resource load, which can all be measured within the node. Buffer use and CPU load are measured entirely within a node. The number of transmissions per incoming packets can also be measured at the node. The nature of this measurement requires, however, an exchange of information between adjacent nodes. It requires the use of the transmit count parameter in the received packet, which is written in by the sending node. Similarly, packet identity numbers and similar fields must be used. It also requires the estimate of noise error probability for packets, which provided by the parameter selection algorithm in which adjacent nodes participate. However, if adjacent nodes can only be either inoperational or operating correctly, the information available for functioning links is always correct. If an output link disappears, estimates of the packet error probabilities in the link are not possible, but ration computations continue updating the link ration with the current value, until the link is terminated by the link

[4] A counterexample would be a star topology with many, very long arms leading to congestion at the hub.

up/down protocol. Thus ration computations are localized at a node and are not be disrupted by simple failures in adjacent nodes or neighboring links.

Since the existence of acknowledgements is a necessity for correctly functioning paths, rations values back-propagate to sources, provided that a functioning path exists between them and the node updating the rations and that flow exists from the source through this path. Flow rations are not allowed to be reduced to zero, and so flow exists as long as the source has something to send and the feedback loop between sources and bottlenecks is present. Moreover, the higher the flow the more frequent the acknowledgements on the path and the faster the backpropagation. In this sense, the performance is tuned to the most critical flow control situations. Thus, as long as surviving paths remain between sources and bottlenecks, there exists a feedback path to backpropagate ration information from bottleneck to sources.

Finally, enforcing flow rations is done independently by each source node, and independently for each flow from that node: problems in failed paths do not affect congestion control for functioning paths. Thus, the algorithm operation over functioning paths is not affected by simple node or link failures. Such failures, however, can cause congestion in neighboring nodes as forwarding delays and retransmissions increase (ultimately towards infinity) around them. This congestion is detected, as in any normal congestion situation, by the neighboring nodes affected and congestion control follows. The fact that the algorithm strives to implements max-min fairness implies that oftentimes the effects of the problems are confined only to sources sharing the newly developed bottlenecks. This contributes to graceful degradation of throughput in the network as a whole[5]. This discussion argues for the robustness of the algorithm in the face of simple node and link failures. Notice that it assumes that other network layers remain functioning properly in surviving nodes.

We now consider a situation where nodes are allowed to behave incorrectly. Ration values computed at a node can be affected by incorrectly functioning neighbors. Since computation of the packet error probability resulting from noise depends on information provided by neighboring nodes on the link in question, erroneous information may lead to overestimation or underestimation of the correct rations. In the SURAP 4 environment, the problem is alleviated because steady state packet error probabilities, due to noise, on input links should not be believed if noticeably greater than 0.1. The main problem would be in erroneous ration information from downstream nodes. If an incorrectly high ration value is received, flow fails to be throttled. However, the problem eventually spreads to correctly functioning nodes which then take over the flow control procedure. If incorrectly low ration values are received, flow may be throttled unnecessarily. This presents a problem, but is confined only to flows whose paths traverse the misbehaving nodes. Similar situations can arise. When misbehaving nodes fail to communicate any rations at all, the ration computation algorithm works with an obsolete value of rations, which either is correct or falls into one of the two previous cases of incorrect values. When packets arrive with false information, for example transmit count, computation of the correct transceiver or output link utilizations is compromised. Another serious problem occurs if a source disobeys its path ration values. The source gradually floods the network with packets, compromising throughput. In summary, misbehaving nodes affect paths traversing them and path traversing nodes affected by the spreading congestion. The confinement properties of the algorithm are thus an important robustness aspect of the algorithm. Ultimately, the congestion control algorithm has to rely on the security architecture protocols to prevent incorrect behavior by nodes for a prolonged time and on the correctness of the software resident in the network nodes.

[5] The *K*-Buffering algorithm also helps to confine forwarding problems to paths experiencing that problem

5. CONCLUSIONS

5.1 SUMMARY

We have presented the design for a distributed rate-based congestion control algorithm suitable for packet radio networks, and we have described the behavior of the algorithm as observed through analytical and simulation results. Although our congestion control algorithm has been designed for the SURAP 4 architecture of the Survivable Adaptive Packet Radio Network under research at BBN, this algorithm can be easily modified to operate in other packet radio architectures.

Traffic flow rates are controlled through resource rations, which are computed by the nodes. A resource ration is the maximum rate at which any traffic flow may use a resource with minimal risk of congestion. Each node monitors the use of its resources – transceiver, buffers, CPU, and output link throughput as constrained by channel noise – and periodically computes resource rations based on the measured utilizations. Our simulation results have suggested that the transceiver and the buffers are more likely than the CPU and the output link throughput to become the saturated node resources under heavy load, and thus are more likely to influence traffic flow rates.

At the source nodes, flow rations control the rates of the flows to the various destinations. A flow ration is the minimum resource ration along the path to a given destination. All nodes use a backpropagation mechanism to distribute ration information to the sources as follows. In response to the receipt of a data packet, the receiving node returns to the sender in the acknowledgement its flow ration to the packet's destination. The sender then uses this information plus its own resource ration information to compute its flow ration to the given destination. This process is repeated all along the source-destination path.

5.2 CONCLUSIONS

Feasibility of Approach Our simulation results confirm the feasibility of the rate-based resource use congestion control approach for packet radio networks. Even though the algorithm presented is designed specifically for the SURAP 4 architecture, the approach can be readily generalized to other packet radio network architectures, for example SURAP 1. The main traits of the congestion control algorithm are:

1. Fast rate of convergence. Rapid convergence is possible for the following three reasons: the algorithm anticipates rather than reacts to congestion; the ration adjustment ratio results in exponential ration decrease under heavy resource load; and the backpropagation mechanism quickly and efficiently distributes ration information to the source nodes.

2. Fairness. The algorithm heuristically achieves max-min fair flow control in several important situations; exceptions are related to channel access dynamics. We characterized fairness and its exceptions in Section 4.5.

3. Robustness. The algorithm can tolerate link and node failures.

In our simulated networks, congestion was for the most part controlled by transceiver rations and by buffer rations. The transceiver is typically the bottleneck for large-packet traffic, while buffers are typically the bottleneck for small-packet traffic. Buffer bottlenecks are not so much a function of the buffer capacity of the nodes as they are of the maximum delay per node deemed acceptable in the network. The CPU can become a bottleneck for small-packet traffic in nodes with several neighbors; however, transceiver contention dominates as the number of neighbors increases. Output link throughput only becomes a bottleneck when link quality is so poor that error correcting bits and packet retransmissions severely restrict the maximum throughput of the link.

As part of our investigation of congestion control, we assessed the effect of the link-layer protocols on the effectiveness of the congestion control algorithm. Most of the link-layer protocols contain flow control mechanisms for reducing the burden on local saturated resources. However, these protocols have no global knowledge of network flows, and so are not able to provide max-min fair flow control. The link-layer protocols operate on a shorter time scale than the congestion control algorithm, and hence may react more quickly to congestion. With link-layer flow control, congestion tends to move back from the bottleneck resource to the source nodes. Thus, the link-layer protocols may inadvertently shield the original source of congestion from the congestion control algorithm and hence prevent it from exercising fair flow control. The link-layer protocol parameters must be chosen carefully, so that the flow control provided by these protocols serves to alleviate transient resource saturation but does not interfere with the long-term fair flow control provided by congestion control.

Rate of Convergence We have not attempted an intrinsic characterization of the advantage afforded by the resource utilization approach with respect to speed of convergence. From the simulation results presented in Section 4, we infer that the approach is a promising way to achieve a relatively fast rate of convergence to steady state flow control. We discussed in Section 4.6 a characterization of the rate of convergence in the event of sudden onrush of traffic. For large networks of many small traffic flows, the rate of convergence has a logarithmic dependence on the number of sources sharing the critical bottlenecks.

Fairness The basis for max-min fair flow in the congestion control algorithm is the equal sharing of bottleneck resources by the flows using them. Node monitor resources to detect bottlenecks and indirectly deliver this information to the sources through backpropagation. Fairness can be achieved in many network flow situations. However, unfair flow can occur as a result of the interaction among neighboring nodes caused by channel access contention. The problem arises when a bottleneck node and another less utilized node contend for a common downstream neighbor. Throttled traffic through the bottleneck frees resources at the common neighbor which can lead to increased flow from the other contending node. The result may be excessive contention at the bottleneck or a low service rate in its output link, causing further throttling by the bottleneck and higher flows through the contending node. Without a greater, more complex exchange of information among adjacent nodes it is hard to prevent this problem.

Robustness Several factors combine to make the proposed algorithm tolerant to link and node problems. First, the distributed and localized nature of the node ration computations means that failure of a link or a node does not disrupt ration computations at adjacent nodes. Second, the fact that congestion problems spread first to adjacent nodes means the normally functioning neighbors will be the first to detect congestion caused by a failed node or link and will exercise control of their ration values. Third, the distributed backpropagation mechanism ensures that ration information will be returned to all reachable sources. The use of acknowledgements as the vehicle for sending ration information guarantees the survivability of congestion control operation, in so far as acknowledgements are a precondition for network existence and provides rapid feedback to the sources as traffic rates increase, precisely when timely feedback becomes most critical.

Congestion control can be adversely affected by problems such as false ration values, disobeyed flow rations, and other forms of aberrant node behavior. We have not designed into the congestion control algorithm any specific mechanisms to protect against these types of failures. The congestion control algorithm relies on the ability of the security architecture protocols to prevent malicious intervention and on the correctness of the software implemented to provide correct node operation.

Overhead The proposed congestion control algorithm has been designed to consume a minimal amount of transmission bandwidth and node processor cycles and memory. Nevertheless, it does incur some costs. The per packet operations include resource load monitoring, backpropagation, and throttling.

For most of its resources, a node updates its measure of resource load for each packet received. Usually, this entails updating the stored value of cumulative load and the current packet count. This is neither a computationally intensive nor a memory intensive task, but it is performed for each packet received.

For every packet correctly received from a neighbor, a node returns an acknowledgement containing the bits-per-second and packets-per-second flow rations for the packet's destination. There is no computation required, and the ration values require only three bytes of space (assuming an uncompressed representation) in the acknowledgement. However, the node must keep track of flow rations to those destinations used by all traffic passing through the node. In the worst case, it must store flow rations for all network destinations, even if its own hosts need to communicate to only a single destination. For every acknowledgement received, a node must update its flow rations to the destination of the original packet, using the flow rations contained in the acknowledgement. This requires comparing the existing values to the received values and storing the smaller of the two.

Each souce node must keep track of throttling information for each destination with which its hosts communicate. The throttling information indicates when the node may next accept from a host a packet for a given destination. In the worst case, a node must maintain throttling information for every destination, but typically the requirements will be much less. The timing mechanism required for throttling will be invoked for every packet sent from the node's hosts and may be the most computationally significant part of the whole congestion control algorithm.

The only other congestion control overhead is the periodic computation of resource rations. Although the ration computations require several arithmetic operations, they occur at intervals on the order of several seconds, and hence will contribute only a small processing load.

5.3 FURTHER RESEARCH

In designing a congestion control algorithm for packet radio networks, the most difficult part has been providing a max-min fair flow control mechanism, as we discussed in Section 4.5. Our quest for a simple mechanism for achieving max-min fair flow control has branched in three directions. First, we are searching for better models of node operation which allow nodes to calculate feasible throughput values on a dynamic basis. Second, we are looking into the use of source-supplied flow rate information as a way to speed convergence and achieve fairness. Third, we are experimenting with a ration control algorithm, based on stochastic search, that uses measured throughput and observed flow patterns to determine whether to increase or decrease ration values.

We believe that two other important directions for research related to congestion control are its implementation in a multiple-path routing environment and its implementation in a hierarchical network environment. Multiple parallel source-destination paths provide a way to improve the survivability of a network and a way to share load aong resources. Work is needed to determine the most efficient way to enforce rations at the source nodes and the interaction among traffic flowing along the different paths. The definition of max-min fairness will need to be extended to this new environment.

Hierarchical networks are a solution to the algorithm complexities occurring in large networks. The essence of the approach is to divide the network into a hierarchy of subnetworks so that information from one subnetwork can be hidden from another subnetwork. Thus, protocol interaction among subnetworks is simplified, due to the reduced complexity of the connecting hierarchy. The fact that information is hidden from subnetwork to subnetwork may imply a fundamental limitation to the goal of max-min fairness in the congestion control algorithm, which is based on global knowledge of network operation as opposed to local knowledge. The congestion control related issues are the amount and nature of the information that will have to be exchanged at the different hierarchical levels to guarantee proper operation if possible.

Appendix A. NETWORK PARAMETER VALUES

The table below details the parameter values used in the simulations discussed in Section 4. The names used correspond to the parameter names used in the software (underscores would replace blank spaces between words, for example, min_pkt_length instead of min pkt length). Names in bold faces refer to network design parameters relevant to congestion control. They should be considered as our design recommendation, subject to the considerations of network size and hardware discussed in Section 4.

Some of the values below deserve a comment. The value *input limit* refers to the maximum number of buffers reserved for the attached host device. This was mentioned in the explanation of link level protocols in Section 2. The value *output limit* is not relevant to the current design. It is present in the simulator for the purpose of research. It refers to the maximum number of buffers in use allowed for any output link queue, and does not exist in the current link level protocol suite for SURAP 3. Since 2 buffers out of 11 are reserved for the host only 9 buffers are available for through traffic for any output link. For the experiments presented in this document the implementation of *output limit* does not make a difference. It would make a small difference in situations where a host were forwarding traffic through an output link and other through traffic were also using the same output link. In this case implementation of *output limit*= 9 would limit to nine the total number of buffers in use for that queue while the link level protocols in Section 2 would allow up to eleven buffers to be used in the same situation. The values *min pps ration* and *min bps ration* were used in the network experiments run. We in fact recommend values dependent on the maximum network size expected. For a network with hundreds of nodes the values recommended are *min pps ration* = 0.01 and *min bps ration* = 1. These values have the disadvantage of allowing very long inter-packet times to be enforced while throttling sources, which implies a possibly long time to find out that bottlenecks have disappeared, but guarantees throttling action when hundreds of end-to-end traffic sources share a bottleneck.

Table A.1: SIMULATION PARAMETER VALUES.

NAME	*VALUE*	*DESCRIPTION*
seed	5	random generator seed
min pkt length	224 bits	minimum data packet size
max pkt length	3000 bits	maximum data packet size
ack length	100 bits	ack packet size
buffers	11	buffers in node
input limit	2.pkts	fec check packet queue limit
thread limit	9 pkts	K in K-Buferring
output limit	9 pkts	output link queue limit
spacing delay	0.02 sec	initialization spacing value
mos duration	0.018 sec	sender MOS interval
fec delay	0.001 sec	fec check processing time
ack delay	0.001 sec	ack reception processing time
send ack delay	0.013 sec	ack creation processing time
forward delay	0.004 sec	data packet processing time
max pps ration	30.0	maximum pps ration
min pps ration	1.0	minimum pps ration
max bps ration	60000.0	maximum bps ration
min bps ration	100.0	minimum bps ration
node ration int	2 secs	ration measurement interval
pps ration int	2	buffer measurement interval scaling factor
path ration int	0.5 sec	flow ration update interval
re x thresh	1.5	transceiver threshold
busy thresh	6.0	buffer threshold
target cpu	0.9	cpu threshold
ration quantum bps	1.2	bps ration increment term
ration quantum pps	1.2	pps ration increment term

Bibliography

[1] E. Baker and S. Eisner. *Congestion Control Software Design Description.* Technical Report, BBN Technical Report. Cambridge,MA, 1986.

[2] (Hazeltine Corporation). *Low Cost Packet Radio (LPR) IF radio User's Guide.* Technical Report, DARPA SRNDOC 14, Arlington, VA, August 1985.

[3] J. Escobar. *Radio-Parameter Selection Algorithm for Receiver-Directed Packet-Radio Networks (SRNTN–73).* Technical Report, BBN Technical Report 7172, Cambridge,MA, December 1989.

[4] J. Jubin and J. Tornow. The DARPA packet radio network protocols. *Proc. of IEEE Special Issue on Packet Radio Networks*, January 1987.

[5] G. Lauer. Survivable packet radio network protocols. *NATO, Shape technical center conference*, III(1):R.1–R.15, March 1989.

[6] M. Leib and J. Zavgren. *High-Throughput, Survivable Protocols for CDMA Packet Radio Networks (SRNTN–74).* Technical Report, BBN,Technical Report 7173, Cambridge,MA, March 1990.

[7] J. Zavgren. The moment-of-silence channel-access algorithm. *Proceedings of IEEE MILCOM89*, 2:20.3.1–20.3.7, October 1989.

[8] J. Zavgren and G. Lauer. The performance improvement from receiver-directed transmissions in packet-radio networks. *IEEE Tactical Communications Conference*, 65–72, May 88.